|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство образования и науки Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)

КАФЕДРА Системы обработки информации и управления (ИУ5)

**ЧЕРНОВИК**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

***Исследование векторного представления метаграфов***

Студент ИУ5-41 **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** А.А. Фадеев

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Ю.Е. Гапанюк

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Нормоконтролер **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2022 г.*

**РЕФЕРАТ**

Представленная выпускная квалификационная работа магистра посвящена исследованию векторного представления метаграфовой модели знаний, анализу возможностей и определение подходов для решения задачи вложения метаграфа, используя существующие алгоритмы эмбеддинга плоских графов, а также использования результатов эмбеддинга для задачи кластеризации элементов метаграфа.

Актуальность работы подтверждается следующими моментами: отсутствие методик для проведения процедуры вложения для метаграфовой модели данных и отсутствие векторного представления метаграфа как такового, что часто делает невозможным применение метаграфа как входных данных для моделей машинного и глубокого обучения.

В данной работе автор уделяет внимание построению алгоритма преобразования метаграфовой модели к модели плоского графа, для которого возможно применение существующих алгоритмов эмбеддинга графов. При этом необходимо соблюдать изоморфность преобразования моделей данных без потери исходной информации, а также корректную интерпретацию иерархических элементов метаграфовой модели во вложенном пространстве. Также автор исследовал возможность получения векторных представлений из метаграфа и применения вложенных пространств на примере задачи кластеризации элементов метаграфовой модели.

Недостатком данной работы можно считать недостаточное количество разнообразных алгоритмов эмбеддинга плоских графов, а также недостаточное количество исследований, сценариев применения векторных пространств, полученных при операции эмбеддинга метаграфов.

# Анализ предметной области

## Метаграфовая модель данных

С каждым годом модели на основе сложных сетей развиваются и применяются во всё более разных условиях, в самых разных областях, не только в математике и информатике, но и биологии, а также социологии. Одним из примеров таких моделей являются гиперсетевая модель, детально рассмотренная в работах профессора Попкова [1].

Одним из главных свойств, которые используются в сложных сетях, является свойство эмерджентности. Эмерджентность - это свойство систем приобретать свойства, не присущие её компонентам в отдельности. Другими словами, эмерджентность - это свойство несводимости свойств системы к сумме свойств входящих в неё элементов. Как выразил эту мысль в работе [2] К.В. Анохин: «формализм гиперсетей обобщает сети и гиперграфы, давая аппарат, необходимый для отображения феноменов эмерджентности в многоуровневых системах». Эмерджентность в метаграфах обеспечивается за счет использования метавершин и может применяться на одном уровне или между уровнями и не обязательно соседними, что делает реализацию принципа эмерджентности более гибкой.

Таким образом, сложные сети и метаграфы, в частности, можно охарактеризовать как «сети с эмерджентностью». Фрагмент такой сети, который состоит их вершин и связей между ними, может выступать как отдельное целое.

На кафедре «Системы обработки информации и управления» МГТУ им. Н.Э. Баумана в рамках данного направления предложена именно метаграфовая модель, как наиболее гибкую. Данную модель предлагается применять как средство для описания сложных сетей [3], как средство для описания семантики и прагматики информационных систем [4], как средство для описания гибридных интеллектуальных информационных систем [5].

В работе [4] была предложена формализованная модель метаграфа, а также рассмотрены основные операции над метаграфами на основе «информационных элементов метаграфа» (ИЭМ). В дальнейшем в работе [6] в качестве внутренней модели представления метаграфа вместо ИЭМ было предложено использование предикатного описания.

## Основные положения метаграфовой модели

Основополагающими работами по теории метаграфов являются работы А. Базу и Р. Блэннинга, которые в 2007 году были обобщены в виде монографии [7]. Модель, предложенная в [7], в дальнейшем была адаптирована для описания различных аспектов интеллектуальных информационных систем и представлена в статьях [5] [8] [9] [10]. В данном разделе кратко рассмотрим формализованную модель метаграфа.

В целях ясности изложения предлагаемого подхода в данном разделе рассматриваются основные положения метаграфовой на основании материалов статей [4][5][6]. Рассмотрим формализованное представление метаграфовой модели:

где – метаграф; *V* – множество вершин метаграфа; *MV* – множество метавершин метаграфа; *E* – множество ребер метаграфа.

Вершина метаграфа характеризуется множеством атрибутов:

где – вершина метаграфа; – атрибут.

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

где – ребро метаграфа; – исходная вершина (метавершина) ребра; – конечная вершина (метавершина) ребра; – признак направленности ребра ( – направленное ребро, – ненаправленное ребро); ­– атрибут.

Фрагмент метаграфа:

,

где – фрагмент метаграфа; – элемент, принадлежащий объединению множеств вершин, метавершин и ребер метаграфа.

Таким образом фрагмент метаграфа в общем виде может содержать произвольные вершины, метавершины и ребра метаграфа без ограничений.

Метавершина метаграфа:

где – метавершина метаграфа, принадлежащая множеству вершин *MV*; – атрибут, – фрагмент метаграфа.

Исходя из вышеописанного, метаверщина обладает свойством вложенности, те включает в себя фрагмент метаграфа, которым также может содержать вершины графа, метавершины либо ребра.

Наличие у метавершин собственных атрибутов и связей с другими вершинами является важной особенностью метаграфов. Это соответствует принципу эмерджентности, то есть приданию понятию нового качества, несводимости понятия к сумме его составных частей. Фактически, как только вводится новое понятие в виде метавершины, оно «получает право» на собственные свойства, связи и т.д., так как в соответствии с принципом эмерджентности новое понятие обладает новым качеством и не может быть сведено к подграфу базовых понятий. Таким образом, метаграф можно охарактеризовать как «граф с эмерджентностью», то есть фрагмент графа, состоящий из вершин и связей, может выступать как отдельное целое.

Пример описания небольшого метаграфа показан на рисунке 1.1.



Рис. 1.1– Пример описания метаграфа

Данный метаграф содержит вершины, метавершины и ребра. На рис. 1 показаны четыре метавершины: mv1, mv2, mv3 и mv4. Метавершина mv1 включает вершины v1, v2, v3 и связывающие их ребра e1, e2, e3. Метавершина mv2 включает вершины v4, v5 и связывающее их ребро e6. Ребра e4, e5 являются примерами ребер, соединяющих вершины v2-v4 и v3-v5, включенные в различные метавершины mv1 и mv2. Ребро e7 является примером ребра, соединяющего метавершины mv1 и mv2. Ребро e8 является примером ребра, соединяющего вершину v2 и метавершину mv2. Метавершина mv3 включает метавершины mv2 и mv4, вершины v2, v3 и ребро e2 из метавершины mv1 а также ребра e4, e5, e8. Метавершина mv4 включает не соединенные ребрами вершины v6 и v7.

В качестве активного элемента метаграфовой модели используется метаграфовые агенты, также рассмотренные в работе [6]. Определим метаграфовой агент следующим образом:

где , , на основе которого выполняются правила агента, т.е. рабочий метаграф. стартовое условие выполнения агента (фрагмент метаграфа, который используется для стартовой проверки правил, или стартовое правило).

Структура правила метаграфового агента:

.

Метаграфовые правила можно разделит на разомкнуты и замкнутые правила. Замкнутые правила меняют правой частью правила фрагмент метаграфа, относящийся к левой части правила. Изменение метаграфа заставляет срабатывать левые части других правил. При этом некорректно разработанные правила могут привести к зацикливанию агента.

Разомкнутые правила не меняют правой частью правила фрагмент метаграфа, относящийся к левой части правила, можно сравнить входной и выходной фрагменты метаграфа.

Таким образом агенты позволяют генерировать фрагменты один на основе другого или изменять метаграф с использованием замкнутых либо разомкнутых правил.

## Представление метаграфа как иерархически организованного множества предикатов

В работе [6] в качестве внутренней модели представления метаграфа вместо ИЭМ предлагается использовать предикатное описание. Классическим примером языка на основе предикатов является язык Пролог, который использует следующую форму предикатного описания: . В работе [6] предлагается использовать расширенную форму предикатного описания: . Данная форма, в дополнение к атомам, может также содержать пары ключ-значение и вложенные предикаты. Различные варианты отображения элементов метаграфовой модели в предикатное описание представлены в таблице 1.1.

Таблица 1.1 – Соответствие метаграфовой модели предикатному описанию

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Фрагмент метаграфа** | **Предикатное описание** |
| 1 |  | Metavertex(Name=  =mv1, v1, v2, v3) |
| 2 |  | Edge(Name=e1, v1, v2) |
| 3 |  | Edge(Name=e1, v1, v2, eo=false) |
| 4 |  | 1. Edge(Name=e1, v1, v2, eo=true)  2. Edge(Name=e1, vS=v1, vE=v2, eo=true) |
| 5 |  | Metavertex(Name=mv2, v1, v2, v3,  Edge (Name=e1, v1, v2),  Edge(Name=e2, v2, v3),  Edge(Name=e3, v1, v3)) |

Продолжение таблицы 1.1

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Фрагмент метаграфа** | **Предикатное описание** |
| 6 |  | Metavertex(Name=mv2, v1, v2, v3, Edge(Name=e1, vS=v1, vE=v2, eo=true), Edge(Name=e2, vS=v2, vE=v3, eo=true), Edge(Name=e3, vS=v1, vE=v3, eo=true)) |
| 7 |  | Attribute(количество, 5) |

В таблице 1.1 (пункт 1) показан пример метавершины , которая содержит 3 вложенных несвязных вершины 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, показанный на таблице 1.1 (пункт 1), является простейшим, поскольку вложенные вершины не связаны друг с другом.

Ребро метаграфа можно рассматривать как частный случай метавершины, которая включает в себя две вершины: исходную и конечную. Пример для ненаправленного ребра показан на таблице 1.1 (пункт 2). В предикатном описании. Ребро может рассматриваться как предикат, а исходную и конечную вершины как параметры предиката.

Пример ненаправленного ребра, который полностью соответствует формальному определению, показан на таблице 1.1 (пункт 3). В этом случае метавершина помечается соответствующей аннотацией направленности, а предикат добавляется именованный параметр, соответствующий признаку направленности.

Пример направленного ребра показан на таблице 1.1 (пункт 4). В предлагаемой модели все параметры могут быть именованными. В пункте 4 именованные параметры соответствуют параметрам из формального определения ребер метаграфа.

Метавершина, содержащая вершины и ребра, может быть представлена с использованием предикатов высших порядков. На таблице 1.1 (пункт 5) имеется пример метавершины с ненаправленными ребрами, а пункт 6 – пример метавершины с направленными ребрами. В соответствии с описанием метаграфа, предикаты, отвечающие за вершины и ребра вложены на одном уровне в предикат метавершины.

Фрагмент метаграфа может содержать произвольное количество вершин, метавершин либо ребер.

Любой атрибут метаграфа может быть представлен как частный случай метавершины куда вложены две связные вершины, где в одной вершине содержится ключ параметра, а во второй вершине его значение. На таблице 1.1 (пункт 7) представлен атрибут, содержащий целое число. Для Атрибута используется имя предиката «Attribute». Значения атрибутов могут иметь строковые, числовые и другие значения. Однако, ключи только строковые.

Предикатное описание позволяет представить метаграфовую модель в текстовом виде, и составные части предикатного описания достаточно «атомарны». Но предикатное описание не дает информации о том, какие действия можно выполнять над элементами метаграфовой модели.

Рассмотрим пример описания метаграфа, показанный на рисунке 1.2, в виде иерархического предикатного представления.



Рисунок 1.2 – Иерархическое предикатное представление метаграфа

Иерархическая зависимость между предикатами представляет плоский граф. На рисунке 1.2 предикаты показаны в виде вершин плоского графа. Предикаты, соответствующие ребрам, обозначены пунктиром. Стрелки направлены от предикатов верхнего уровня к предикатам нижнего уровня.

Граф, представленный на рисунке 1.2, является стратифицированным. Нижний уровень содержит предикаты-вершины. Второй уровень включает предикаты, соответствующие связям между элементами первого уровня. Более высокие уровни содержат метавершины и связи между ними.

Необходимо отметить, что количество уровней стратифицированного графа можно рассматривать как косвенную характеристику сложности, соответствующей метаграфовой модели.

Предикатное представление метавершин и ребер не имеет существенных различий. Так, второй уровень графа содержит шесть ребер и метавершину mv4. Ребро можно считать частным случаем метавершины, которое может включать только две вложенных вершины и обладает признаком (атрибутом) направленности.

Этот вывод может показаться парадоксальным с точки зрения классической теории графов, где вершины и ребра считаются принципиально различными объектами. Но с точки зрения системного подхода в этом нет ничего удивительного. Можно рассматривать ребро как совокупность двух вложенных вершин, при этом факт наличия ребра между вершинами обозначает появление эмерджентности. Две вершины, соединенные ребром, представляют собой отдельный объект, более высоко организованный чем просто две отдельные вершины.

Тогда метавершина представляет собой эмерджентность более высокого порядка, которая включает как исходные вершины, так и соединяющие их ребра, и, возможно, метавершины нижнего уровня. При этом вершины, ребра и метавершины представлены отдельными предикатами.

Элементарные вершины представляют собой ноль-эмерджентность, то есть рассматриваются как атомарные элементы, внутренняя организация которых в модели не учитывается. Рассмотренный подход представляет собой основу метаграфового исчисления.

Единственной структурой данных исчисления является вершина-предикат. Необходимо отметить, что у вершины-предиката очень прозрачная семантика – это множество вложенных вершин-предикатов нижнего уровня.

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

# Разработка метода эмбеддинга метаграфов

## Особенности эмбеддинга метаграфов

Метаграфовая модель является новой и неизученной моделью данных. Поэтому на данный момент не существует методов эмбеддинга метаграфов, однако в статье [13] предложен метод изоморфного преобразования метаграфа в n-дольный граф, который в свою очередь возможно преобразовать в вектора с помощью алгоритмов приведенных в [11] [12]. Автор в данной работе предлагает исследовать возможность эмбеддинга метаграфа через его плоский n-дольный граф.

## Преобразования метаграфа в многодольный граф

Определение метаграфа не дает полной информации о том, каким образом хранить метаграфы в информационной системе. Модель метаграфа является достаточно высокоуровневой моделью представления данных и знаний. С точки зрения СУБД такую модель можно рассматривать как «логическую» модель. Но для использования метаграфов в информационных системах необходимо также предложить варианты «физической» модели, с помощью которой метаграфовые данные можно хранить в СУБД.

Рассмотрим представление метавершины в виде графа, показанное на рисунке 1.2.

На основании рисунка 2.1 метавершину можно представить в виде комбинации обычной вершины и набора связей от данной вершины к другим вершинам (или метавершинами) и ребрам. Поскольку метавершина содержит связи не только с другими вершинами, но и с ребрами, то ребро в этом случае удобно рассматривать в виде вершины отдельного класса, которая имеет неименованные связи с вершинами (метавершинами).

Таким образом, вершины и ребра образуют двудольный граф с той поправкой, что если вершина является метавершиной, то она содержит дополнительные иерархические связи с другими вершинами (метавершинами) и ребрами. Если выделить метавершины в отдельный класс, то фактически граф становится трехдольным.



Рисунок 2.1 – Метавершина (показана пунктиром) в виде фрагмента метаграфа (а) и ее представление в виде графа (б)

Если метаграф является аннотированным, то аннотации могут быть заданы для вершин (метавершин) и ребер, которые показаны на рис. 11б в виде окружностей и ромбов, но не могут быть заданы для «примитивных» связей, которые показаны на рис. 11б в виде линий.

Фактически рисунок 2.1 показывает вариант преобразования метаграфа к плоскому графу. На основании рисунка 2.1б можно сделать вывод о том, что СУБД на основе графовой модели могут использоваться для хранения метаграфов, но при этом метавершины и ребра должны быть искусственно преобразованы к вершинам и соединены неименованными связями.

Таким образом, в случае представления вершин (метавершин) и ребер с помощью двудольного (трехдольного) графа, вершины и ребра представляют собой относительно похожие структуры.

## Алгоритм преобразования метаграфа в многодольный 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:

,

где – набор вершин графа; – набор ребер графа и модель метаграфа как:

,

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

Тогда плоский графа может быть рассмотрен как 3-дольный граф метаграфа :

Набор вершин графа может мыть разделено на 3 независимых набора Существует 3 изоморфных преобразования между вершинами метаграфа, метавершинами, ребрами и соответствующими множествами:

Набор содержит информацию о связях между вершинами, метавершинами и ребрами в начальном метаграфе.

С точки зрения многодольных графов, нет разницы граф ориентированный или нет, потому что ребра в графе представляются как вершины и знак ориентирования ребра в может быть отображен как атрибут новой вершины ребра в . Мы рассматриваем отношения между сущностями как специальную сущность более высокого уровня, которая включает сущности нижнего уровня.

Алгоритм отображения метаграфовой модели в модель 3-дольного графа можно представить с помощью алгоритма 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. Также возможна реализация плоского графа при помощи списков смежности или sparse-матриц. Изначально, для удобства весь метаграф является начальной метавершиной .

Алгоритм преобразования модели метаграфа в 3-дольный граф, состоящий из 2 методов TRS и MCP, представлен ниже:

Процедура TRS():

* Пройдем по всем элементам , где :
  + Если не находится в списке , :
  + Если является метавершиной , то вызвать процедуру TRS();
  + Добавить в список .
  + Процедура MCP( – множество):
* Пройдем по всем элементам списка , где :
  + Если является метавершиной :
    - Для каждого элемента метавершины , заполнить матрицу и , где .
  + Если является ребром , то заполнить матрицу , где

В результате работы алгоритма метаграф изображенный на рисунке 2.2 преобразуется в 3-дольный граф, изображенный на рисунке 2.3.

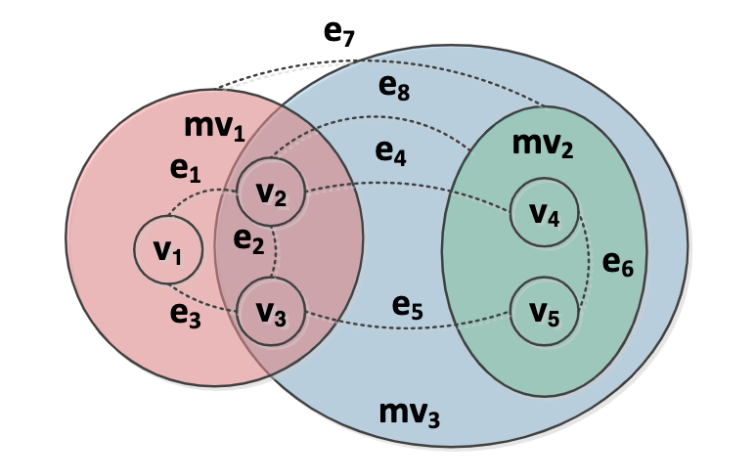


Рисунок 2.2 – Пример отображаемого метаграфа

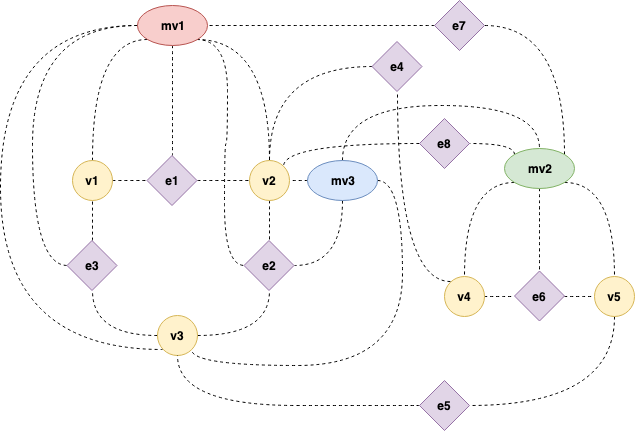


Рисунок 2.3 – 3-дольный граф, полученный при отображении метаграфа

Как видно из рисунка 4, плоский граф также состоит 3 типов вершин: вершин-метавершин ; вершин-вершин ; вершин-ребер Все вершины соединены ребрами, для которых не имеет значение их направленность, так как для метаграфа значение направленности вершин установлено в атрибутах ребер, которые сохраняются в вершинах-ребрах при преобразовании в 3-дольный граф.

Метавершины соединены ребрами со своими вложенными элементами исходного метаграфа, а остальные элементы соединены в соответствии с общей топологией.

## Эмбеддинг плоских графов

Изложение механизмов эмбеддинга графов в данном и следующих разделах в основном опираются на обзоры [11],[12], а также на статьи, посвященные конкретным алгоритмам эмбеддинга.

Хотя графовая аналитика практична и важна, большинство существующих методов графической аналитики страдают от высоких вычислительных и пространственных затрат. Много исследований было посвящено эффективному проведению дорогостоящей графовой аналитики. Примеры включают в себя инфраструктуры обработки данных распределенного графа (например, GraphX [14], GraphLab [15]), другие компактное хранилище графов, которое ускоряет ввод-вывод и вычислительные затраты и т. д. [16].

В дополнение, вложение графа предоставляет эффективный способ решения задачи аналитики графов. В частности, вложение графа преобразует граф в низкоразмерное пространство, в котором сохраняется информация о графе. Представляя граф в виде (или набора) низкоразмерных векторов, графовые алгоритмы могут быть эффективно вычислены. Результатом встраивания графа является низкоразмерный вектор, представляющий часть графа (или целый граф). На рисунке 2.4 показан пример встраивания графа в двумерное пространство. То есть, в соответствии с различными потребностями, мы можем представить узел как низкоразмерный вектор.

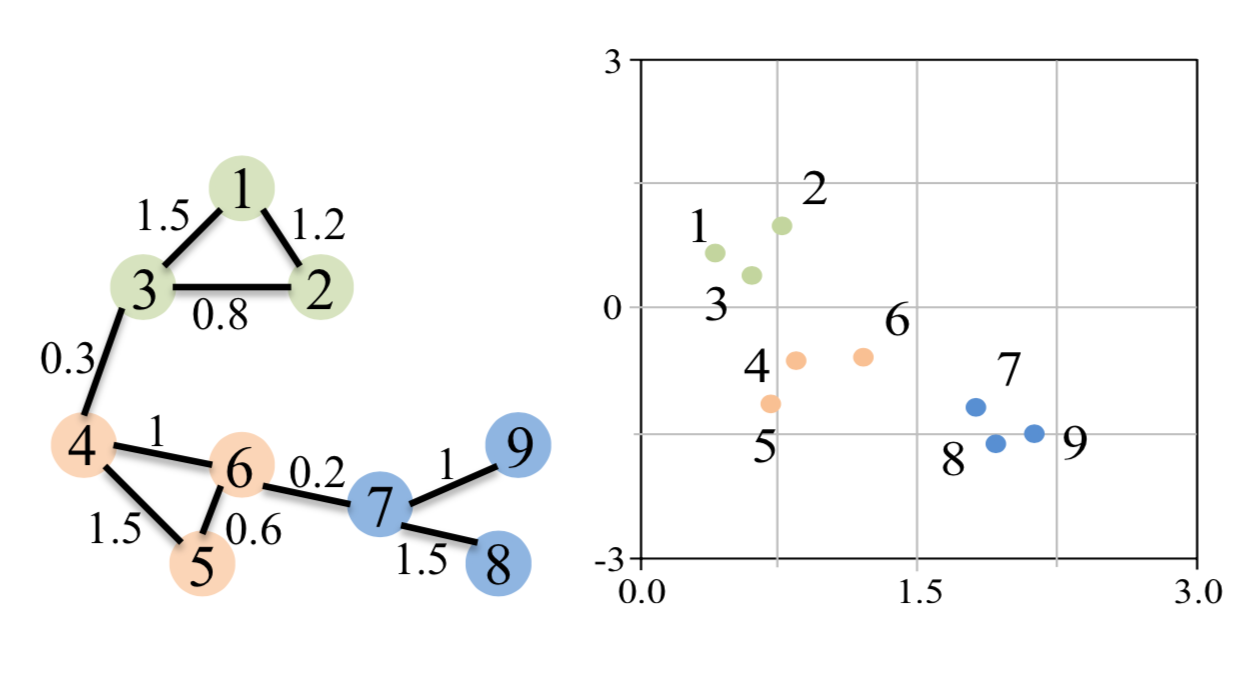


Рисунок 2.4 – Пример встраивания вершин графа в двумерное пространство

В начале 2000-х алгоритмы встраивания графов были в основном разработаны для уменьшения высокой размерности нереляционных данных, предполагая, что данные лежат в низкоразмерном многообразии. Учитывая набор нереляционных признаков многомерных данных, граф сходства строится на основе парного сходства признаков. Затем каждый узел в графе внедряется в низкоразмерное пространство, где связанные узлы расположены ближе друг к другу.

С 2010 года, с распространением графа в различных областях, исследования по внедрению графа начали принимать граф в качестве входных данных и использовать вспомогательную информацию для облегчения эмбеддинга.

В частности, вложение графа направлено на представление графа в виде низкоразмерных векторов, в то время как структура графа сохраняются. С одной стороны, графическая аналитика направлена на извлечение полезной информации из данных графа. С другой стороны, при обучении получают представления данных, которые облегчают извлечение полезной информации при построении классификаторов или других предикторов. Внедрение графа или эмбеддинг фокусируется на изучении низкоразмерных представлений.

Вложение графов в низкоразмерные пространства не является тривиальной задачей. Проблемы внедрения графа зависят от постановки задачи, которая состоит из ввода и вывода эмбеддинга. наиболее распространенный тип эмбеддинга - это эмбеддинг узла, которое представляет близкие узлы в виде схожих векторов. Эмбеддинг узла может принести пользу задачам, связанным с узлами, таким как классификация узлов, кластеризация узлов и т. д. Однако в некоторых случаях задачи могут быть связаны с более высокой гранулярностью графа, например, пары узлов, подграф, весь граф. Следовательно, первая задача в терминах встраивания выходных данных – найти подходящий выходной тип эмбеддинга для применения интереса. Мы классифицируем четыре типа выходных данных эмбеддинга графов: эмбеддинг узлов, эмбеддинг ребер, гибридный эмбеддинг и эмбеддинг целого графа. Разные детализации выходных данных имеют разные критерии для «хорошего» встраивания и сталкиваются с различными проблемами. Например, хорошее вложение узла сохраняет сходство с соседними узлами в векторном пространстве. Напротив, хорошее вложение целого графа представляет целый граф как вектор, так что уровень графа так что сходство на уровне графа сохраняется. Пример встраивания ребер. подграфа и графа представлен на рисунках 2.5 - 2.7.

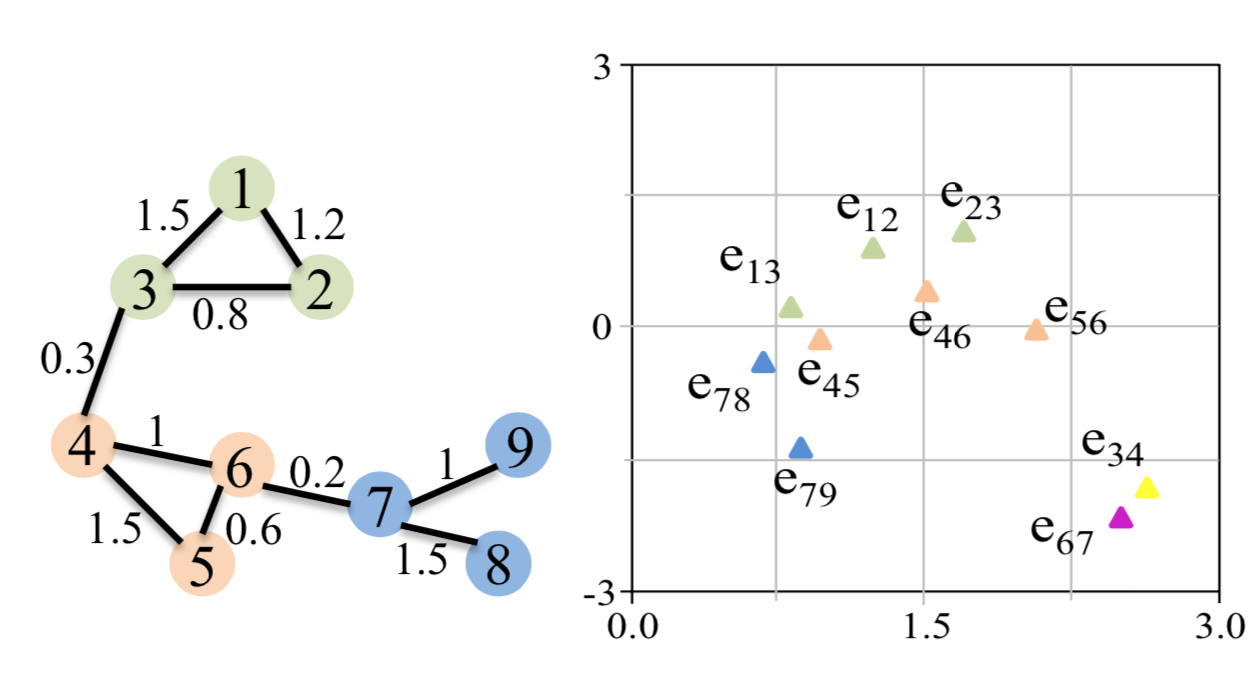
****

Рисунок 2.5 – Пример встраивания ребер в двумерное пространство

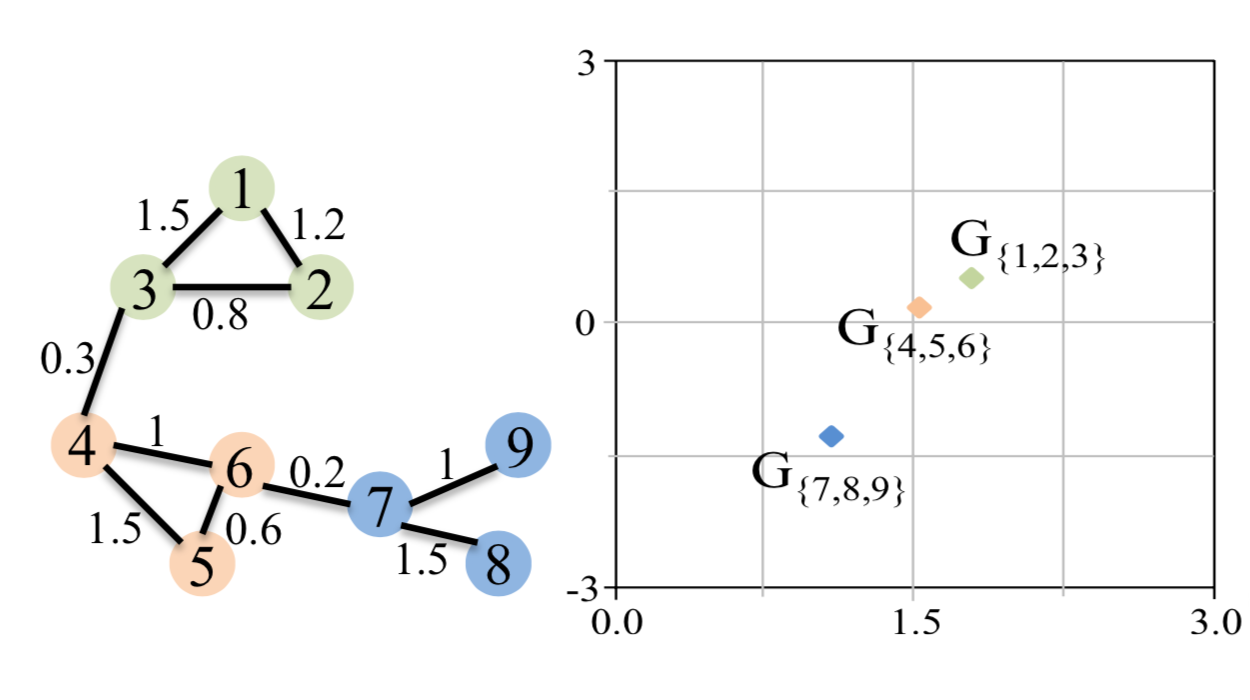


Рисунок 2.6 – Пример встраивания подграфа в двумерное пространство

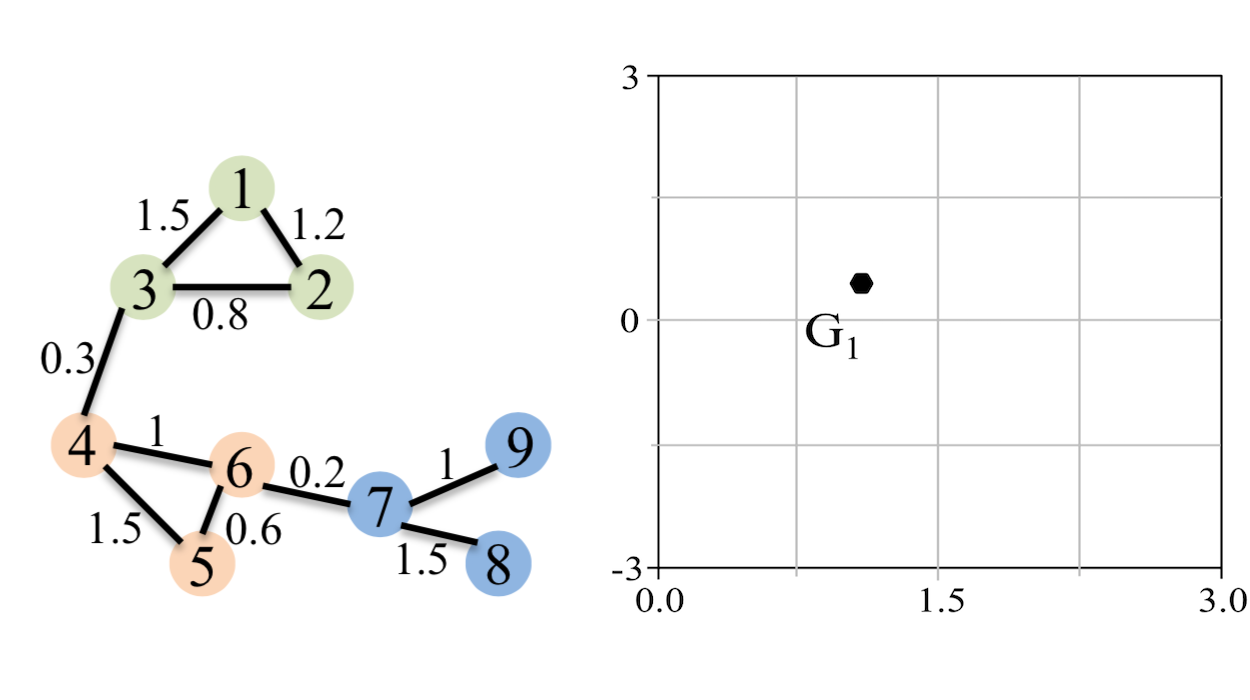


Рисунок 2.7 – Пример встраивания графа в двумерное пространство

Плоские графы (сети) очень большой размерности. Такие сети могут включать миллионы и более вершин. Ребра, соединяющие вершины, могут быть ненаправленными или направленными. Иногда используется модель мультиграфа, в этом случае две вершины могут соединяются не одним, а несколькими ребрами. Именно такую модель в литературе чаще всего называют «сложной сетью». Важно, что в рамках данного подхода «сложная сеть» остается плоским графом (мультиграфом).

Вторая трактовка сложного графа – графы, в которых используется сложное (комплексное) описание вершин, ребер и/или их расположения. Часто в таких моделях используется не плоский, а пространственный вариант расположения вершин и ребер. Именно подобный подход может быть наиболее полезен при описании сложных моделей данных и знаний. На сегодняшний день наиболее широко известны три подобных модели: гиперграф, гиперсеть и метаграф. В данной работе мы будем говорить только о метаграфовой модели.

В настоящее время во многих прикладных задачах необходимо рассматривать сложные графы очень большой размерности (включающие миллионы вершин). Обработка плоского графа или мультиграфа является чрезвычайно сложной задачей как по памяти, так и по вычислительным ресурсам. В случае обработки сложных графов задача еще более усложняется за счет комплексного описания моделей.

Одним из наиболее современных подходов к решению задачи обработки графов большой размерности является отказ от традиционного представления графа в виде множества вершин и множества ребер непосредственно при обработке графа, что показано на рисунке 2.8.

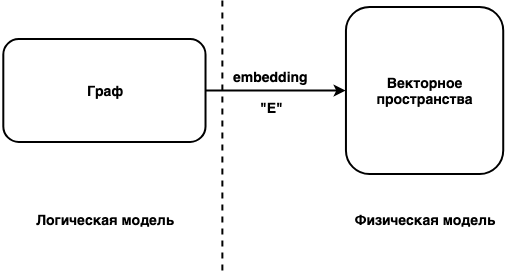


Рисунок 2.8 – Модель графа с точки зрения его обработки

Модель сложного графа с точки зрения его обработки рассматривается на двух уровнях. По аналогии с моделями данных реляционных СУБД, эти уровни можно назвать «логической» и «физической» моделью сложного графа.

Логическая модель сложного графа – это традиционная модель плоского графа или мультиграфа, включающая множества вершин и ребер.

В настоящее время в качестве физической модели представления графов 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 “E” на рис. 5). В англоязычной литературе для обозначения такого преобразования традиционно используется термин «embedding», то есть «встраивание» графа в векторное пространство.

Операция «векторного представления» широко используется в машинном обучении, в частности, для обработки текстов. Хорошо известны алгоритмы векторного представления для текстов: Word2Vec, Glove и др. Часто англоязычный термин не переводят и в русском варианте используют термин «эмбеддинг», как синоним термина «векторного представления». Далее в статье мы также будем использовать термин «эмбеддинг».

Необходимо отметить, что в отличие от реляционной СУБД, где преобразование «логической» модели в «физическую» является относительно несложной операцией, эмбеддинг графа является достаточно сложной процедурой, как правило, связанной с решением задач оптимизации.

Необходимо также отметить, что в отличие от реляционной СУБД, где преобразование «логической» модели в «физическую» является однозначным отображением, эмбеддинг графа можно рассматривать как типичную связь «один-ко-многим», на основе одной логической модели с помощью различных методов можно сформировать различные результирующие векторные пространства физической модели. Полученные физические модели, как правило, оптимизированы для выполнения конкретных алгоритмов над графами. В зависимости от задачи, может производиться эмбеддинг только вершин графа, комбинированный эмбеддинг вершин и ребер графа, параллельный эмбеддинг вершин и ребер графа в различные векторные пространства.

Значительная часть наиболее простых техник эмбеддинга 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.

## Входные данные

Входными данными для вложения графов является сам граф. А точнее его матрица смежности. Алгоритмы эмбеддинга используют матрицу смежности для встраивания графа. Используются такие метрики как близость 1-го порядка и близость 2-го порядка. Если граф не взвешен, то матрица смежности имеет одинаковые значения для связей, обычно 1. Если же она взвешена, то в ячейку определённой связи ставится вес этого ребра.

В данной работе мы разделяем эмбеддинг на 2 категории однородные графы и многодольные графы. В экспериментальной части мы работаем только с неориентированным и невзвешенным 3-дольным графом. То есть используем близость 2-го порядка.

## Однородные графы

Первой категорией входного графа является однородный граф, в котором узлы и ребра принадлежат одному типу. Однородный граф может быть далее классифицирован как взвешенный (или направленный) и невзвешенный (или ненаправленный) граф, как пример, показанный на рисунке 2.9.

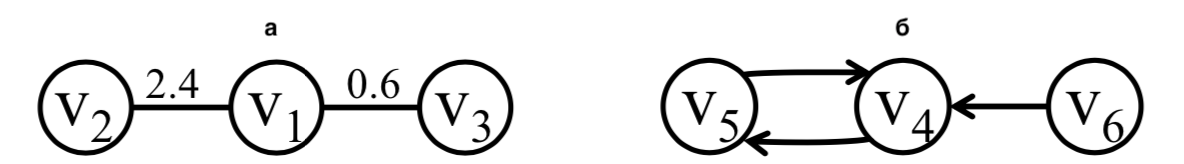


Рисунок 2.9 – Пример взвешенного и направленного графа

Неориентированный и невзвешенный однородный граф – это самая базовая конфигурация ввода в граф. Ряд исследований работают в этом направлении, например, [17], [18], [19], [20], [21]. Они одинаково обрабатывают все узлы и ребра, поскольку доступна только основная структурная информация входного графа. Интуитивно понятно, что веса и направления ребер предоставляют больше информации о графике и могут помочь более точно представить график во встроенном пространстве. Например, на рис. 11 (а) , должен быть встроен ближе к , чем к , потому что вес ребра выше. Точно так же на рисунке 11 (б) должно быть встроено ближе к , чем , так как и соединены в обоих направлениях. Приведенная выше информация теряется в невзвешенном и ненаправленном графике. Заметив преимущества использования веса и направления ребер графа, сообщество по исследователей начинает исследовать взвешенный и/или ориентированный граф. Некоторые из них фокусируются только на одном свойстве, то есть либо вес ребра, либо направление ребра. С одной стороны, взвешенный граф рассматривается в [22], [23], [24], [25], [26], [27]. Узлы, соединенные взвешенными ребрами c большим значением, встроены ближе друг к другу. Однако эти работы все еще ограничены неориентированными графами.

С другой стороны, некоторые работы различают направления ребер во время процесса встраивания (эмбеддинга) и сохраняют информацию о направлении во векторном пространстве. Одним из примеров ориентированного графа является граф социальной сети, например, [26]. Каждый пользователь имеет и подписку, и подписку с другими пользователями. Однако информация о весе недоступна для социальных ссылок пользователей. Недавно был предложен более общий алгоритм вложения графов, в котором рассматриваются как весовые, так и направленные свойства. Другими словами, эти алгоритмы [27], [28], [29] могут обрабатывать как направленный, так и ненаправленный, а также взвешенный и невзвешенный граф. При вложении возникаем задача, а именно: как отобразить разнообразие моделей соединения, наблюдаемые на графах? Поскольку в однородных графах доступна только структурная информация, проблема встраивания однородных графов заключается в том, как сохранить эти шаблоны связности, наблюдаемые во входных графах во время встраивания.

## Многодольные графы

Второй тип входных данных – многодольный граф. Как было сказано выше многодольные графы имеют разные типы (доли) вершин. Пример многодольного графа представлен на рисунке 2.10.

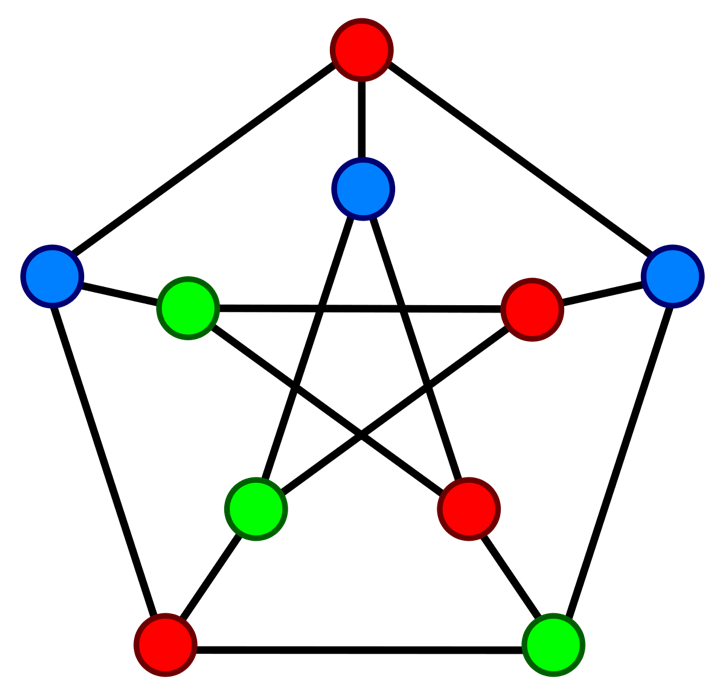


Рисунок 2.10 – Пример многодольного графа

На рисунке 2.10 представлен многодольный граф, а именно трёхдольный так как разные имеем вершины условно трёх типов: «зеленый», «синий» и «красный». Вершины в многодольных имеют разный семантическое значение и принадлежат к разным множествам вершин. В роли каждого из типов могут быть, соответственно, метавершины, вершины и рёбра.

Для многодольных графов множество сценариев применения.

Сайты сообщества вопросов и ответов (cQA). cQA - это краудсорсинговый сервис в Интернете, который позволяет пользователям размещать на веб-сайте вопросы, на которые затем отвечают другие пользователи [30]. Интуитивно понятно, что в графе cQA есть разные типы узлов, например, вопрос, ответ, пользователь.

Мультимедийные сети. Мультимедийная сеть – это сеть, содержащая мультимедийные данные. Например, в [31] и [32] рассмотрены графы, содержащие два типа узлов: изображение, текст и три типа ссылок: совместное появление изображения-изображения, текста-текста и изображения-текста. [33] обрабатывает социальное взаимодействие с пользовательским узлом и узлом изображения. Он использует ссылки на пользовательские изображения для встраивания пользователей и изображений в одно и то же пространство, чтобы их можно было напрямую сравнивать для получения рекомендаций по изображениям. В [34] рассматривается график кликов, который содержит изображения и текстовые запросы. Ребро запроса изображения указывает на щелчок изображения при заданном запросе, где количество кликов служит весом края.

Графы знаний. В графе знаний сущности (узлы) и отношения (ребра) обычно бывают разных типов. Например, в графе знаний о фильмах, построенном из Freebase [35], типами сущностей могут быть «режиссер», «актер», «фильм» и т. д. Типы отношений могут быть «производить», «направлять», «действовать в». Много усилий было посвящено внедрению графов знаний (например, [36], [37], [38]).

Целью данной работы является представить модель сложной сети метаграфа в виде многодольного графа и провести эмбеддинг вершин этого графа. Сложностью здесь является то, как исследовать глобальную согласованность между различными типами объектов и как справляться с дисбалансами объектов, принадлежащих к различным типам? Различные типы объектов (например, узлы, ребра) встраиваются в одно и то же пространство при внедрении разнородного графа. Как исследовать глобальную согласованность между ними ­ проблема. Более того, может существовать дисбаланс между объектами разных типов. Эта асимметрия данных должна учитываться при эмбеддинге.

## Выходные данные

Результатом внедрения графа является набор низкоразмерных векторов, представляющих часть графа. Разделяют выходные данные вложения графов на четыре категории: вложение узлов, вложение ребер, гибридное вложение и вложение целого графа. Встраивание узлов может принести пользу широкому кругу задач анализа графов, связанных с узлами. Представляя каждый узел как вектор, связанные с узлом задачи, такие как кластеризация узлов, классификация узлов, могут быть эффективно выполнены с точки зрения времени и затраченных ресурсов. Однако задачи аналитики графа не всегда находятся на уровне узла. В некоторых сценариях задачи могут быть связаны с более высокой гранулярностью графа, такого как пары узлов, подграф или даже целый граф. Следовательно, первая проблема с точки зрения внедрения вывода заключается в том, как найти подходящий тип вывода, который соответствует потребностям конкретной прикладной задачи.

Так как при преобразовании метаграфа в 3-дольный граф мы имеем 3 типа вершин и ненаправленные связи, которые служат для поддержания изоморфности с исходным метаграфом. Ребром в этом графе также является вершина определенного типа, также, как и метавершина и вершина. В данной работе исследуется ненаправленный метаграф, поэтому соответствующий 3-дольны граф также будет ненаправленным. Поэтому стоит задача вложения вершин всех 3 типов в единое векторное пространство.

В качестве наиболее распространенной вида вывода вложения является вложение узла. Вложение узла представляет каждый узел как вектор в низкоразмерном пространстве. Узлы, которые «близки» в графе, вложены, чтобы иметь аналогичные векторные представления. Различия между различными методами вложения графов заключаются в том, как они определяют «близость» между двумя узлами.

Введем два определения:

Близость первого порядка между вершиной и вершиной является вес . Пара вершин более «схожи» если они соединены ребром с большим значением. Обозначим близость первого порядка между вершиной и вершиной как . Тогда – близость первого порядка между вершиной и остальными вершинами. Например для графа представленного на рисунке 2.11, пусть Тогда, исходя из рисунка 2.11 и определения выше .

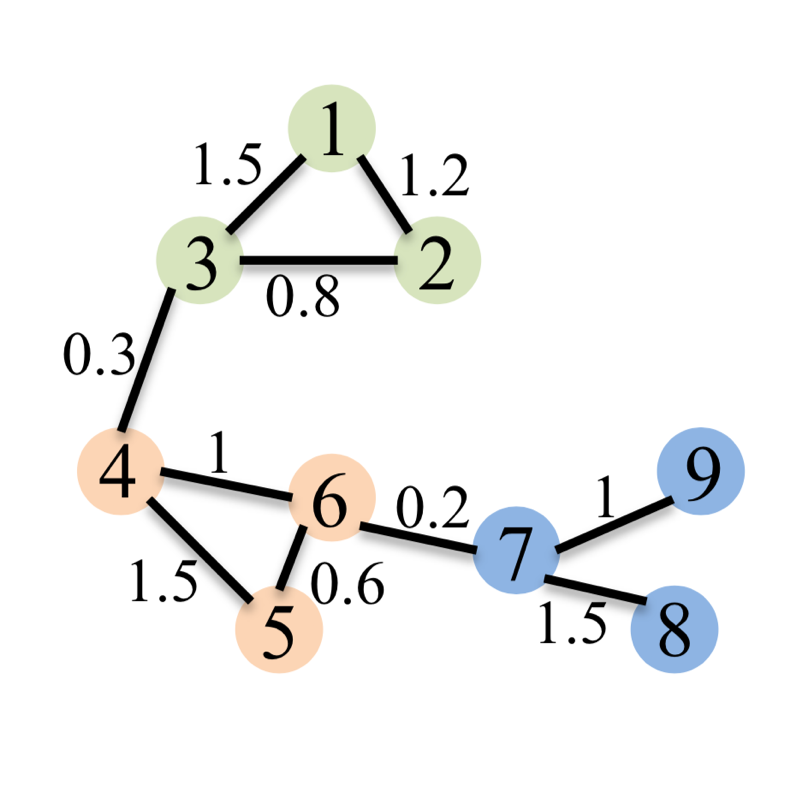


Рисунок 2.11 – Пример плоского графа

Близость второго порядка сравнивает сходство структур соседних узлов. Чем больше схожи «окрестности» двух узлов, тем больше значение близости второго порядка между ними. Близость второго порядка между вершиной и вершиной является схожестью между соседними вершинами вершины и соседней вершиной вершины . Исходя из рисунка 13 , . Воспользуемся косинусным расстоянием.

Две эти метрики используются для расчета подобия попарных узлов.

## Алгоритмы эмбеддинга

В этом разделе мы классифицируем методы эмбеддинга графов на основе используемых методов. Как правило, вложение графа направлено на представление графа в низкоразмерном пространстве, которое сохраняет как можно больше информации о свойствах графа. Различия между различными алгоритмами вложения графов заключаются в том, как они определяют свойство графа, которое будет сохранено. Разные алгоритмы имеют различное понимание сходства узлов и способов их сохранения во векторном пространстве. Далее будет показано несколько техник вложения графов, а также о том, как они количественно определяют свойства графа и решают проблему эмбеддинга графа.

В большинстве случаев входные данные представляют собой график, построенный из нереляционных элементов данных больших размеров и выводом является набор векторов узлов. Таким образом, проблема вложения графов может рассматриваться как задача уменьшения размерности, сохраняющая структуру, которая предполагает, что входные данные лежат в низкоразмерном многообразии. Существует два типа вложения графов на основе факторизации матрицы. Одним из них является собственные карты Лапласа (Graph Laplacian eigenmaps), а другим – факторизация матрицы смежности узлов (Node Proximity Matrix Factorization).

Эмбеддинг на основе матричной факторизации представляет связи графа в форме матрицы и разлагает эту матрицу для получения вложения узла [39]. Матрицы, используемые для представления соединений, включают в себя: матрицу смежности узлов, матрицу Лапласа, матрицу вероятностей перехода узлов, а также другие. Для неструктурированных матриц можно использовать методы градиентного спуска, чтобы получить вложение за линейное время.

Глубокое обучение (DL) показало выдающуюся производительность в широком спектре областей исследований, таких как компьютерное зрение, языковое моделирование и т. д. Вложение графов на основе DL применяет модели DL к графам. Эти модели являются либо прямым применением из других областей, либо новой моделью нейронной сети, специально разработанной для встраивания графов. Входные данные – это либо пути, взятые из графа, либо самого граф. Одним из самых популярных алгоритмов является node2vec.

## Собственные карты Лапласа (Graph Laplacian Eigenmaps)

Свойства графа можно интерпретировать как сходство парных узлов, которые можно оформить в виде матрицы смежности. Для алгоритма Graph Laplacian Eigenmaps оптимальное вложение может быть получено с помощью функции ниже [40] [43]:

Пусть дано n – объектов , алгоритм вложения стремится найти такие , где . В последние десятилетия для решения этой проблемы было предложено множество алгоритмов. Несмотря на различные цели этих алгоритмов, они могут быть хорошо интерпретированы в общей структуре вложения графов.

Учитывая граф G с n вершинами, каждая из которых представляет точку данных. Пусть W – симметричная матрица с весом , Представляющую вес ребра соседних вершин . G и W могут быть охарактеризовывать определенные статистические или геометрические свойства данных. Цель вложения графа - представить каждую вершину графа как низкоразмерный вектор, сохраняющий сходства между парами вершин, где сходство измеряется ребром вес. Чем больше вес ребра, тем больше «схожесть между вершинами графа».

Пусть набор векторов который представляет каждый из вершин графа в низкоразмерном пространстве. Оптимальный набор пытается оптимизировать функцию:

где элемент матрица смежности, вес между вершиной и вершиной .

Оптимизация функции происходит под соответствующим ограничение. Эта целевая функция (1) влечет за собой серьезные штрафы, если соседние вершины i и j отображаются далеко друг от друга. Поэтому сведение к минимуму является попыткой убедиться, что если вершины i и j «близки», то также близки [41]. Поэтому мы имеем:

где – Лапласиан графа [42]. диагональная матрица где . Чем больше значение , тем более значение .

В итоге задача минимизации состоим из минимизации функции:

где ограничение устраняет фактор произвольного масштабирования во вложении.

Вышеупомянутое вложение графа является трансдуктивным, поскольку оно может встраивать только те узлы, которые существуют в обучающем наборе. На практике может также потребоваться встраивание новых узлов, которые не были замечены в обучении. Одним из решений является разработка линейной функции:

которая может предоставлять вложение пока предоставляются элементы узлов. Следовательно, для индуктивного вложения графа, при котором возможно вложение не только вершин из тренировочной выборки, уравнение 2 преобразуется в следующую целевую функцию:

Так же как и в уравнении 3, добавляется ограничение уравнение 4 преобразуется в уравнение:

в которой необходимо найти оптимальное значение .

Оптимальными являются собственные векторы с максимальными собственными значениями при решении уравнения

Различные алгоритмы такие как MDS [44], Isomap [45], LE [46], LLP [47], AgLLP [49], ARE [48], SR [40]. Эти алгоритмы используют этот метод, однако с другим значением . Две матрицы W и D играют существенную роль в этом алгоритмах вложения графа. Выбор этих двух матриц могут быть очень гибкими. Были предприняты некоторые попытки [50], [51] обобщить существующие методы вложения графов, основанные на собственных картах Лапласа, с использованием общей структуры.

Первоначальное исследование MDS [40] непосредственно приняло евклидово расстояние между двумя векторами признаков и как , используется, чтобы найти оптимальное вложение . MDS не учитывает окрестность узлов, т. e. Любая пара обучающих экземпляров рассматривается как связанная.

Последующие исследования, например, [45], [46], [52], [53], преодолевают эту проблему, сначала построив граф k-ближайшего соседа (KNN) из данных. Каждый узел связан только со своими верхними k-подобными соседями. После этого для вычисления матрицы подобия W, используются разные методы, чтобы сохранить как можно больше желаемых свойств графа. Некоторые более продвинутые модели разработаны недавно. Например, AgLPP [49] вводит якорный граф, чтобы значительно повысить эффективность более ранней модели факторизации матрицы LPP.

LGRM [54] изучает модель локальной регрессии, чтобы понять структуру графа и термин глобальной регрессии для экстраполяции данных вне выборки. Наконец, в отличие от предыдущих работ по сохранению локальной геометрии, LSE [55] использует локальную сплайн-регрессию для сохранения глобальной геометрии.

В некоторых других работах предполагается, что W не может быть сконструирован путем простого перечисления парных связей узлов. Вместо этого они применяют полуопределенное программирование (SDP) для обучения W. Полуопределенное программирование [56] стремится найти внутреннюю матрицу произведений, которая максимизирует попарные расстояния между любыми двумя входами, которые не связаны в графе, сохраняя при этом расстояния ближайших соседей. MVU [57] строит такую матрицу, а затем применяет MDS [44] к изученной внутренней матрицы произведений.

## Факторизация матрицы смежности узлов (Node Proximity Matrix Factorization)

Близость узла может быть аппроксимирована в низкоразмерном пространстве с использованием матричной факторизации. Целью сохранения близости узлов является минимизация потери аппроксимации.

Имеем матрицу близости узлов W, целью является минимизация выражения:

,

где – вложение узла, а Y c ∈ R | V | × d – вложение для узлов контекста [58]. 5 стремится найти оптимальное ранговое приближение матрицы близости W, где d – размерность векторов. Одно из популярных решений – применить разложение по сингулярным числам (SVD) к W [59]:

где являются сингулярными значениями, отсортированными по убыванию и являются сингулярными векторами для . Оптимальное вложение получается с использованием наибольших d сингулярных значений и соответствующих сингулярных векторов следующим образом:

В зависимости от того, сохраняется ли асимметричное свойство или нет, вложение узла может быть либо [58], [60], или объединение и то есть [61]. Существуют другие решения для уравнения 6, таких как регуляризованная факторизация гауссовой матрицы [62], матричная факторизация низкого ранга [63], и добавление других регуляризаторов для обеспечения соблюдения больших ограничений [64].

Матричная факторизация в основном используется для вложения графа, построенного из не связанных данных для встраивания узлов, что является типичной постановкой задач Лапласиана о собственных карт. Матричная факторизация также используется для встраивания однородных графов [60].

## DeepWalk и Node2Vec

Глубокое обучение (Deep Learning) показало выдающуюся производительность в широком спектре областей исследований, таких как компьютерное зрение, языковое моделирование и т. д. Встраивание графов на основе глубокого обучения применяет слови модели к графам. Эти модели являются либо адаптацией из других областей, либо новой моделью нейронной сети, специально разработанной для встраивания графовых данных.

В первой категории встраивания графа, основанного на глубоком обучении, граф представляется в виде набора случайных путей. Методы глубокого обучения затем применяются к путям для получения вложения, который сохраняет свойства графа, носимые путями.

Большинство алгоритмов машинного обучения основано на использовании признаков, являющихся числами (а также иногда категориями). Однако, далеко не все сущности, которые хотелось бы использовать в качестве признаков, можно легко перевести в числовой вид. Примерами таких объектов являются слова в тексте или вершины в графе. В нашей работе мы рассматривали задачу вложения: как каждой отдельной сущности (в нашем случае вершине графа) сопоставить точку вещественного n-мерного пространства, или, проще говоря, n вещественных чисел, чтобы относительное расположение этих точек лучшим образом отображало структуру графа.

Стоить отметить, что вложение является задачей обучения без учителя, то есть при построении не учитывает известные метки классов, но только переводит вершины в числовые признаки. После этого полученные числовые признаки могут быть использованы для любых целей, в том числе и для применения в алгоритмах классификации и кластеризации.

Одним из основных алгоритмов данного класса является алгоритм DeepWalk [65] и node2Vec [66]. DeepWalk и Node2Vec основаны на алгоритме word2vec из области обработки естественного языка, изначально работающем с последовательностями слов, а потому только адаптируемом для работы с графами.

Метод вложения DeepWalk основан на идее случайных блужданий. Выбрав одну из вершин графа в качестве начальной точки, произвольным образом перемещаемся в одну из ее соседних вершин. Затем повторяем случайное перемещение из вновь выбранной вершины и т.д. Получившаяся последовательность вершин называется реализацией случайного блуждания по графу [67].

Формально, учитывая исходный узел u, мы моделируем случайное блуждание фиксированной длины l. Пусть обозначает i-й узел в прогулке, начиная . Узлы генерируются следующим распределением:

,

где – ненормализованная вероятность перехода между узлами и , а - нормализующая константа.

Простейшим способом перемещения наших случайных блужданий будет выборка следующего узла на основе статического веса ребра. Однако это не позволяет нам учитывать структуру сети и направлять нашу процедуру поиска для изучения различных типов окрестностей сети. Кроме того, случайные блуждания должны учитывать тот факт, что эти понятия эквивалентности не являются конкурирующими или исключительными, и в реальных сетях обычно проявляется смесь обоих. Мы определяем случайное блуждание 2-го порядка с двумя параметрами p и q, которые направляют блуждание: рассмотрим случайное блуждание, которое только что пересекло ребро (t, v) и теперь находится в узле v, рисунок 2.12.

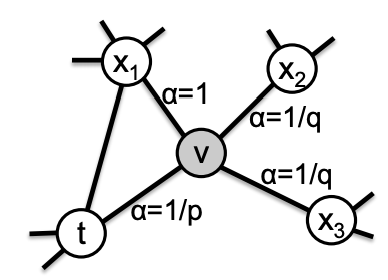


Рисунок 2.12 – Иллюстрация процедуры случайного блуждания в node2vec

Теперь нужно пройтись по следующему шагу, чтобы оценить вероятности перехода на ребрах начиная с . Мы устанавливаем ненормализованную вероятность перехода на = , где

и обозначает кратчайшее расстояние между узлами и . Обратите внимание, что должен быть одним из {0, 1, 2}, и, следовательно, эти два параметра необходимы и достаточны для регулирования прогулкой. Интуитивно понятно, что параметры определяют, насколько быстро проходится обход и покидает окрестность начального узла .

Параметр контролирует вероятность немедленного повторного посещения узла в прогулке. Установка его высокого значения, например более , гарантирует, что мы с меньшей вероятностью выберем уже посещенный узел в следующих двух шагах, если только у следующего узла в обходе не было другого соседа. С другой стороны, если p низкое, например менее это приведет к тому, что прогулка вернется на шаг, и это будет держать прогулку «локальной» близко к начальному узлу , рисунок 14.

Параметр позволяет при поиске различать «внутренние» и «внешние» узлы. Возвращаясь к рисунку 14, если , случайное блуждание смещено к узлам, близким к узлу t.. Напротив, если , прогулка более склонна посещать узлы, которые находятся дальше от узла t. Однако существенным отличием здесь является то, что мы проводим обход в рамках механизма случайного блуждания. Следовательно, отобранные узлы не находятся на строго увеличивающихся расстояниях от заданного исходного узла , но, в свою очередь, мы получаем выгоду от отслеживаемой предварительной обработки и превосходной эффективности выборки случайных блужданий. Обратите внимание, что, устанавливая как функцию предыдущего узла в блуждании , случайные блуждания являются Марковскими процессами 2-го порядка.

Есть несколько преимуществ случайных блужданий. Случайные блуждания эффективны в вычислительном отношении, с точки зрения ресурсов, так и времени. Количество пространства для хранения непосредственных соседей каждого узла в графе равна . Для случайных блужданий 2-го порядка полезно хранить взаимосвязи между соседями каждого узла, что влечет за собой количество пространства где a – средняя степень графа и обычно мала для реального времени. Другим ключевым преимуществом случайных блужданий по сравнению с классическими стратегиями выборки на основе поиска является его временная сложность. В частности, навязывая графическую связность в процессе генерации выборки, случайные обходы предоставляют удобный механизм для увеличения эффективной частоты выборки путем повторного использования выборок в разных узлах источника. Моделируя случайное блуждание длиной , мы можем генерировать выборок для узлов одновременно из-за марковской природы случайного блуждания. Следовательно, для рисунка 14, наша эффективная сложность составляет на выборку. Мы выбираем случайное блуждание длиной , что приводит к , и . Повторное использование выборки может внести некоторую погрешность в общую процедуру. Тем не менее, мы видим, что это значительно повышает эффективность.

Запоминая результат, мы получаем последовательность вершин, которая называется случайным блужданием на графе. Далее каждая вершина интерпретируется как слово, а одно случайное блуждание как предложение. Используя механизм поиска скрытых представлений для слов word2vec [68], методы получают искомое вложение для вершин графа. В данном алгоритме переходы в любую соседнюю вершину равновероятны. Эта технология позволяет составить векторное представление на основе контекстной близости: по аналогии со словами в текстах, встречающиеся в блуждании рядом вершины будут иметь близкие координаты рисунок 2.13.

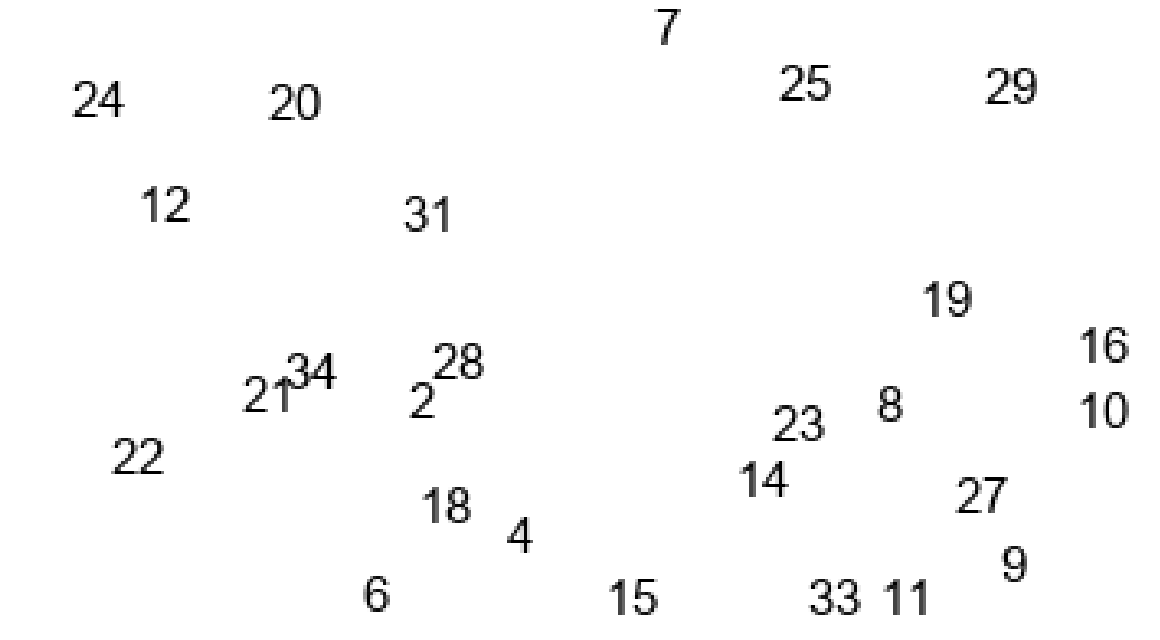


Рисунок 2.13 – Представление графа, полученное с помощью метода DeepWalk

Алгоритм node2Vec является усовершенствованием алгоритма DeepWalk. Основное отличие данного метода от DeepWalk заключается в том, что на каждой итерации блуждания вычисляются вероятности перехода в ту или иную вершину. В зависимости от значений двух параметров p (параметр возврата) и q (параметр углубления), блуждания более склонны к приближению или отдалению относительно начальной вершины [69]. Корректирование этих параметров позволяет сформировать блуждания, наиболее полно отражающие те или иные признаки и структуры каждого отдельного графа [69].

## Structural Deep Network Embedding (SDNE)

Последние достижения в области глубоких нейронных сетей свидетельствуют о том, что они обладают мощными способностями представления [74] и могут генерировать очень полезные представления для многих типов данных. Например, [75] предложил семислойную сверточную нейронную сеть для генерации представлений изображений для классификации. [76] предложили мультимодальную глубокую модель для изучения унифицированных представлений текста в изображении для выполнения задачи поиска кросс-модальности. Однако, насколько известно, было мало работ по глубокому обучению, работающих с сетями, особенно с обучением представлениям сетей. В [77] были использованы ограниченные машины Больцмана для совместной фильтрации. [78] принял глубокий автоэнкодер для кластеризации графов. [79] предложили гетерогенную глубокую модель для встраивания разнородных данных. Наша работа направлена на изучение низкоразмерных структурно-сохраненных сетевых представлений, которые можно использовать для выполнения различных задач. Во-вторых, мы рассматриваем близость первого и второго порядка между вершинами, чтобы сохранить локальную и глобальную структуру сети. Но они сосредоточены только на информации одного порядка.

SDNE предлагаем глубокую модель для встраивания сети, структура которой показана на рисунке 2.14. Чтобы охватить сильно нелинейную сетевую структуру, мы предлагаем глубокую архитектуру, которая состоит из нескольких нелинейных функций отображения чтобы сопоставить входные данные с сильно нелинейным скрытым пространством для охвата структуры сети. Кроме того, для решения проблем сохранения структуры и разреженности модель, использует близость второго и первого порядка. Для каждой вершины мы можем получить ее окрестность. Соответственно, мы проектируем неконтролируемый компонент, чтобы сохранить близость второго порядка, восстанавливая структуру окрестности каждой вершины. Между тем, для небольшой части пар узлов мы можем получить их попарные сходства, то есть близости первого порядка. Поэтому мы разрабатываем контролируемый компонент для использования близости первого порядка в качестве контролируемой информации для уточнения представлений в скрытом пространстве. Совместно оптимизируя их в предлагаемой глубокой модели, SDNE может хорошо сохранять сильно нелинейную локально-глобальную сетевую структуру и устойчива к разреженным сетям.

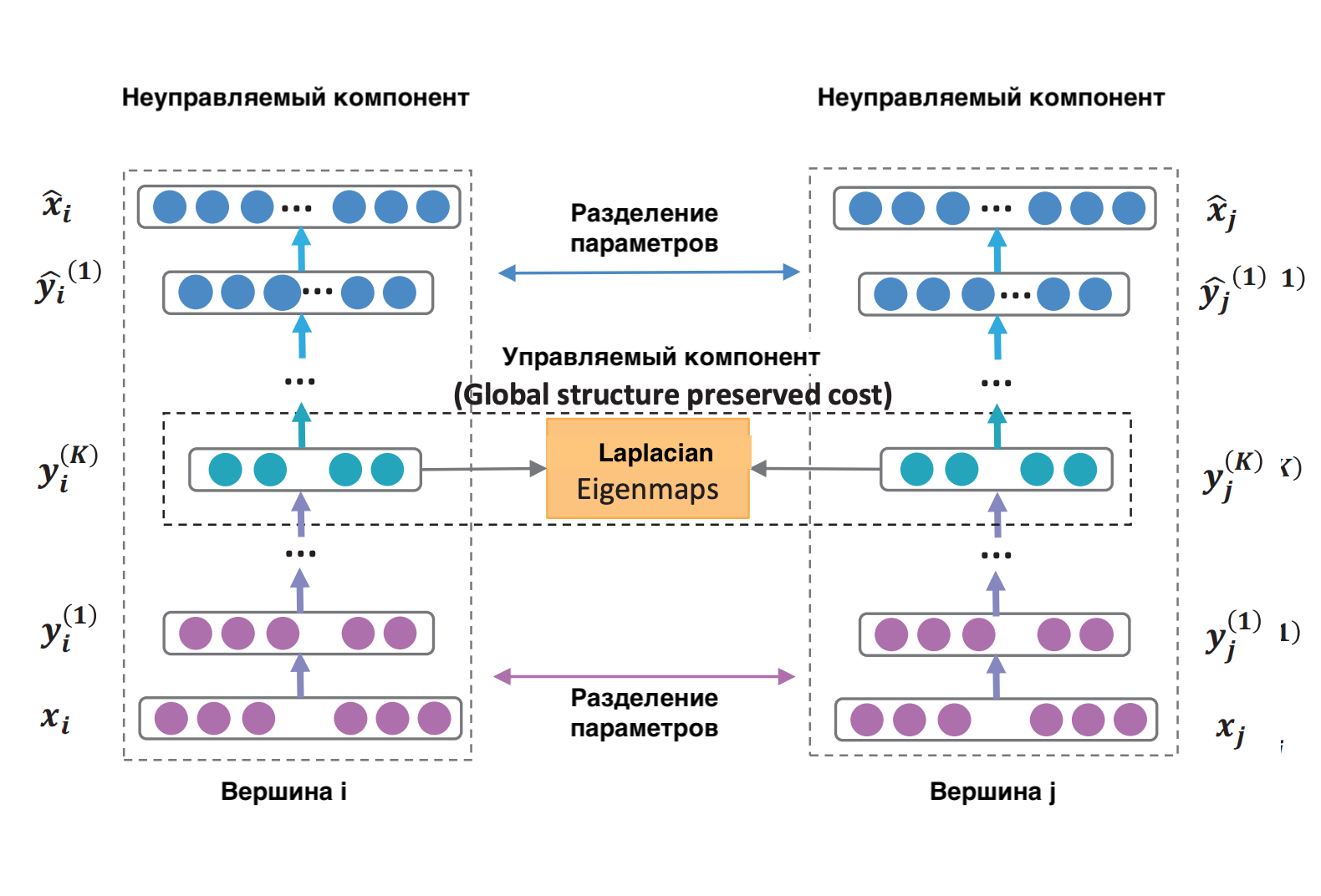


Рисунок 2.14 – Схема глубокой модели SDNE

Прежде чем вводить функции потерь, мы определим некоторые из

термины и обозначения в таблице 2, которые использоваться в работе. Обозначением «^» помечены параметры декодера.

Таблица всех обозначений представлена на таблице 2.2.

Таблица 2.2 – SDNE. Таблица обозначений

|  |  |
| --- | --- |
| **Символ** | **Определение** |
| n | Количество вершин |
| K | Количество слоев в нейросети |
|  | Матрица смежности графа |
|  | Входные и восстановленные данные |
|  | Скрытые представления k-ого уровня |
|  | Матрицы весов k-ого уровня |
|  | Смещения k-ого уровня |
|  | Общие параметры |

Ведем функции потерь для модели. Сначала мы опишем, как неуправляемый компонент использует близость второго порядка для сохранения структуры глобальной сети. Близость второго порядка относится к тому, насколько похожа структура окрестностей пары вершин. Таким образом, для моделирования близости второго порядка требуется моделировать окрестность каждой вершины. Имея граф , мы можем получить ее матрицу смежности, которая содержит экземпляров . Для каждого экземпляра тогда и только тогда, когда существует связь между . Следовательно, описывает структуру окрестности вершины , а S предоставляет информацию о структуре окрестности каждой вершины. С помощью S мы расширяем традиционный глубокий автоэнкодер [23], чтобы сохранить близость второго порядка.

Чтобы рассмотреть возможность автономности, мы кратко рассмотрим ключевую идею глубокого автоэнкодера. Это неконтролируемая (unsupervised) модель, которая состоит из двух частей, то есть энкодера и декодера. Энодер состоит из нескольких нелинейных функций, которые отображают входные данные в пространство представления. Декодер также состоит из множества нелинейных функций, отображающих из пространства представления в пространство восстановления. Затем, учитывая входные данные , скрытые представления для каждого слоя показаны следующим образом:

;

После получения мы можем получить выходной сигнал , перевернув процесс вычисления кодера. Целью автоматического энкодера является минимизация ошибки восстановления выходных данных и входных данных. Функция потерь представлена следующим образом:

.

Хотя минимизация потерь при восстановлении явно не сохраняет сходство между выборками, критерий восстановления может плавно охватывать многообразия данных и, таким образом, сохранять сходство между выборками. Затем, если мы будем использовать матрицу смежности S в качестве входных данных для автоэнкодера, то есть , так как каждый экземпляр характеризует структуру окрестности вершины , процесс восстановления сделает вершины, имеющие схожие соседние структуры, с похожими скрытыми представлениями [81].

Тем не менее, такой процесс реконструкции не может быть непосредственно применен к этой задаче из-за некоторых специфических характеристик сетей. В сетях мы можем наблюдать некоторые ссылки, но одновременно многие законные ссылки не наблюдаются, что означает, что ссылки между вершинами указывают на их сходство, но никакие ссылки не обязательно указывают на их различие. Более того, из-за разреженности сетей число ненулевых элементов в S намного меньше, чем число нулевых элементов. Тогда, если мы напрямую используем S в качестве входных данных для традиционного автоэнкодера, более вероятно восстановить нулевые элементы из S. Однако это не то, что мы хотим. Чтобы решить эту проблему, мы налагаем больше штрафов на ошибку восстановления ненулевых элементов, чем на нулевые элементы. Пересмотренная целевая функция показана следующим образом:

где – произведение Адамара, .

Теперь, используя пересмотренный глубокий автоэнкодер с матрицей смежности S в качестве входных данных, вершины, имеющие похожую структуру окрестностей, будут отображаться рядом в пространстве представлений, что гарантируется критерием реконструкции. Другими словами, неуправляемый компонент модели может сохранить структуру глобальной сети, восстанавливая близость второго порядка между вершинами.

Это необходимо не только для сохранения структуры глобальной сети, но и для охвата локальной структуры. Мы используем первый порядок близости (first order proximity), чтобы обозначить структуру локальной сети. Близость первого порядка может рассматриваться как контролируемая информация, чтобы ограничить сходство скрытых представлений пары вершин. Поэтому мы разрабатываем контролируемый компонент для использования близости первого порядка. Функция потерь для этой цели определена следующим образом:

Целевая функция уравнения. 4 заимствует идею Лапласовских собственных карт [40], рассмотренных выше, которая влечет за собой штраф, когда подобные вершины отображаются далеко в пространстве вложения. В результате модель сохраняет близость первого порядка.

Чтобы сохранить близость первого и второго порядка одновременно, мы предлагаем модель, которая объединяет уравнение. 6 и уравнение 7. Модель минимизируют следующую целевую функцию:

где – нормальный L2 регуляризатор, необходимый для предотвращения обучения и определяющийся следующим образом:

DeepWalk сочетал в себе случайное блуждание для изучения графовых представлений. Несмотря на то, что он эмпирически эффективен, ему не хватает четкой целевой функции, чтобы сохранить структуру сети. Node2Vec имеет явную целевую функцию, которая направлена на одновременное сохранение локальной и глобальной структуры путем сохранения близости как первого, так и второго порядка, что видно из уравнения 8.

# Кластеризация вложенного пространства графа

Во многих прикладных задачах измерять степень сходства объектов существенно проще, чем формировать признаковые описания. Например, гораздо легче сравнить две фотографии и сказать, что они принадлежат одному человеку, чем понять, на основании каких признаков они схожи. Задача классификации объектов на основе их сходства друг с другом, когда принадлежность обучающих объектов каким-либо классам не задаётся, называется задачей кластеризации [80].

Задача кластеризации заключается в следующем. Имеется обучающая выборка и функция расстояния между объектами . Требуется разбить выборку на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из объектов, близких по метрике , а объекты разных кластеров существенно отличались. При этом каждому объекту приписывается метка кластера .

Алгоритм кластеризации – это функция , которая любому объекту ставит в соответствие метку кластера . Множество меток в некоторых случаях известно заранее, однако чаще ставится задача определить оптимальное число кластеров, с точки зрения того или иного критерия качества кластеризации.

Решение задачи кластеризации принципиально неоднозначно, и тому есть несколько причин. Во-первых, не существует однозначно наилучшего критерия качества кластеризации. Известен целый ряд достаточно разумных критериев, а также ряд алгоритмов, не имеющих чётко выраженного критерия, но осуществляющих достаточно разумную кластеризацию «по построению». Все они могут давать разные результаты. Во-вторых, число кластеров, как правило, неизвестно заранее и устанавливается в соответствии с некоторым субъективным критерием. В-третьих, результат кластеризации существенно зависит от метрики ρ, выбор которой, как правило, также субъективен и определяется экспертом. Кластеризация отличается от классификации тем, что метки исходных объектов изначально не заданы, и даже может быть неизвестно само множество . В этом смысле задача кластеризации ещё в большей степени некорректно поставленная, чем задача классификации.

## Цели кластеризации

Цели кластеризации могут быть совершенно различными в зависимости от конкретной прикладной задачи:

1. Понять структуру множества объектов , разбив его на группы схожих объектов. Упростить дальнейшую обработку данных и принятия решений, работая с каждым кластером по отдельности.
2. Сократить объём хранимых данных в случае сверхбольшой выборки Xℓ , оставив по одному наиболее типичному представителю от каждого кластера.
3. Выделить нетипичные объекты, которые не подходят ни к одному из кластеров.

В первом случае число кластеров стараются сделать поменьше. Во втором случае важнее обеспечить высокую степень сходства объектов внутри каждого кластера, а кластеров может быть сколько угодно. В третьем случае наибольший интерес представляют отдельные объекты, не вписывающиеся ни в один из кластеров.

Во всех этих случаях может применяться иерархическая кластеризация, когда крупные кластеры дробятся на более мелкие, те в свою очередь дробятся ещё мельче, и т. д. Результатом является не простое разбиение множества объектов на кластеры, а древообразная иерархическая структура. Вместо номера кластера объект характеризуется перечислением всех кластеров, которым он принадлежит, от крупного к мелкому.

## Метрика коэффициент силуэта (Silhouette score)

Для вычисления метрики используется функция определенная функция. Мы будем использовать эту метрику так как она не требует знания истинных меток кластеров [82].

Пусть:

* *a –* среднее расстояние между текущей точкой и другими точками этого же кластера;
* *b –* среднее расстояние между текущей точкой и другими точками следующего ближайшего кластера.

Тогда коэффициент силуэта для точки (объекта) определяется как:

Силуэтом выборки называется средняя величина силуэта объектов данной выборки. Таким образом, силуэт показывает, насколько среднее расстояние до объектов своего кластера отличается от среднего расстояния до объектов других кластеров. Данная величина лежит в диапазоне [-1;1]. Значения, близкие к -1, соответствуют плохим (разрозненным) кластеризациям, значения, близкие к нулю, говорят о том, что кластеры пересекаются и накладываются друг на друга, значения, близкие к 1, соответствуют "плотным" четко выделенным кластерам. Таким образом, чем больше силуэт, тем более четко выделены кластеры, и они представляют собой компактные, плотно сгруппированные облака точек. С помощью силуэта можно выбирать оптимальное число кластеров (если оно заранее неизвестно) — выбирается число кластеров, максимизирующее значение силуэта. В отличие от предыдущих метрик, силуэт зависит от формы кластеров, и достигает больших значений на более выпуклых кластерах, получаемых с помощью алгоритмов, основанных на восстановлении плотности распределения.

# Экспериментальная часть

## Входной метаграф

В качестве входного метаграфа, который мы будем исследовать, будем использовать неориентированный метаграф изображенный на рисунке 4.1.

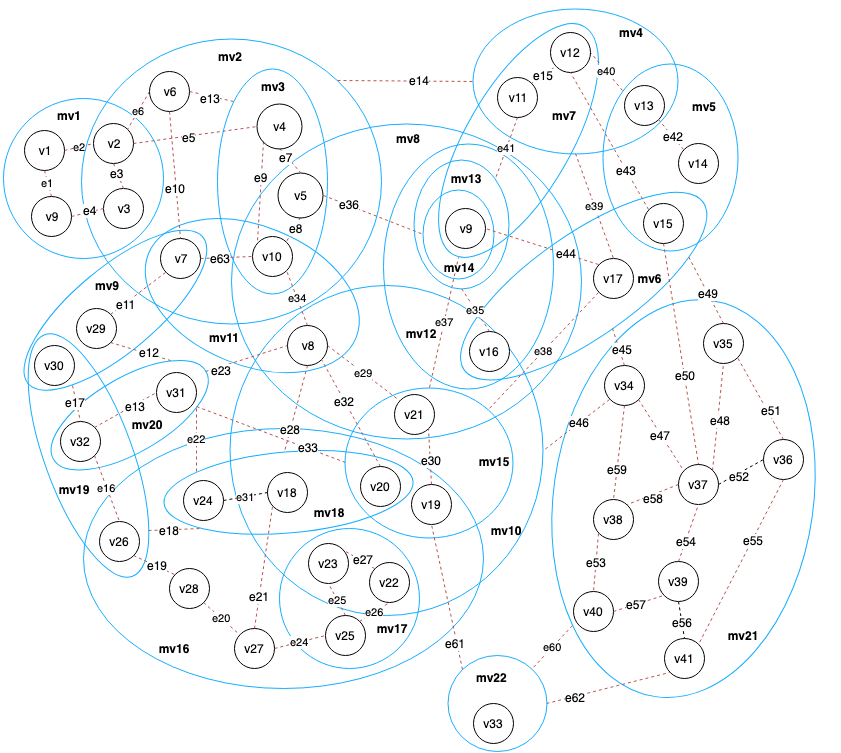


Рисунок 4.1. Входной метаграф.

Этот метаграф состоит из 126 элементов, в число которых входят метавершины, вершины и ребра в соответствие с моделью, представленной в пункте 1. В метаграфе, представленной на рисунке 4.1, имеется 22 метавершины, 41 вершина и 63 ребра.

Метаграф, данной конфигурации был выбран вследствие наглядности модели, присутствием достаточного количества разнообразных связей, наличием иерархических элементов в составе метаграфа, а также наличием метавершин, которые включают в себя разнообразные элементы метаграфа из различных уровней вложения.

Описание всех элементы метаграфа, представленных на рисунке 4.1, отображены в таблице 4.1. Обозначение множеств p – родительские элементы, ch – дочерние; d – исходная вершина ребра, s – конечная вершина ребра. Так как метаграф (рисунок 4.1) неориентированный, d и s являются условными обозначениями.

Таблица 4.1 – Элементы входного метаграфа

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Связи** |
| 1 | v1 | вершина | p{mv1} |
| 2 | v2 | вершина | p{mv1, mv2} |
| 3 | v3 | вершина | p{mv1, mv2} |
| 4 | v4 | вершина | p{mv3, mv2} |
| 5 | v5 | вершина | p{mv1, mv2, mv8} |
| 6 | v6 | вершина | p{ mv2} |
| 7 | v7 | вершина | p{mv2, mv9, mv11} |
| 8 | v8 | вершина | p{mv8, mv10, mv11} |
| 9 | v9 | вершина | p{mv1} |
| 10 | v10 | вершина | p{mv2, mv3, mv11, mv8} |
| 11 | v11 | вершина | p{mv7, mv4} |
| 12 | v12 | вершина | p{mv7, mv4} |
| 13 | v13 | вершина | p{mv4, mv5} |
| 14 | v14 | вершина | p{mv5} |
| 15 | v15 | вершина | p{mv6, mv5} |
| 16 | v16 | вершина | p{mv10, mv8, mv6, mv12} |
| 7 | v17 | вершина | p{mv6} |
| 18 | v18 | вершина | p{mv18} |
| 19 | v19 | вершина | p{mv15, mv16, mv10} |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Связи** |
| 20 | v20 | вершина | p{mv15, mv10, mv18} |
| 21 | v21 | вершина | p{mv15, mv10, mv8} |
| 22 | v22 | вершина | p{mv17, mv10} |
| 23 | v23 | вершина | p{mv17, mv10} |
| 24 | v24 | вершина | p{mv18} |
| 25 | v25 | вершина | p{mv17} |
| 26 | v26 | вершина | p{mv19, mv16} |
| 27 | v27 | вершина | p{mv16} |
| 28 | v28 | вершина | p{mv16} |
| 29 | v29 | вершина | p{mv9} |
| 30 | v30 | вершина | p{mv19, mv9} |
| 31 | v31 | вершина | p{mv20} |
| 32 | v32 | вершина | p{mv19, mv20} |
| 33 | v33 | вершина | p{mv22} |
| 34 | v34 | вершина | p{mv21} |
| 35 | v35 | вершина | p{mv21} |
| 36 | v36 | вершина | p{mv21} |
| 37 | v37 | вершина | p{mv21} |
| 38 | v38 | вершина | p{mv21} |
| 39 | v39 | вершина | p{mv21} |
| 40 | v40 | вершина | p{mv21} |
| 41 | v41 | вершина | p{mv21} |
| 42 | e1 | ребро | d:v1, s:v9 |
| 43 | e2 | ребро | d:v1, s:v2 |
| 44 | e3 | ребро | d:v2, s:v3 |
| 45 | e4 | ребро | d:v3, s:v9 |
| 46 | e5 | ребро | d:v2, s:v4 |
| 47 | e6 | ребро | d:v6, s:v6 |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Связи** |
| 48 | e7 | ребро | d:v4, s:v5 |
| 49 | e8 | ребро | d:v5, s:v10 |
| 50 | e9 | ребро | d:v4, s:v10 |
| 51 | e10 | ребро | d:v6, s:v7 |
| 52 | e11 | ребро | d:v29, s:mv20 |
| 53 | e12 | ребро | d:v16, s:mv13 |
| 54 | e13 | ребро | d:v31, s:v32 |
| 55 | e14 | ребро | d:mv2, s:mv4 |
| 56 | e15 | ребро | d:v11, s:v12 |
| 57 | e16 | ребро | d:v32, s:v26 |
| 58 | e17 | ребро | d:v32, s:mv9 |
| 59 | e18 | ребро | d:mv19, s:mv18 |
| 60 | e19 | ребро | d:v26, s:v28 |
| 61 | e20 | ребро | d:v28, s:v27 |
| 62 | e21 | ребро | d:v27, s:v18 |
| 63 | e22 | ребро | d:mv18, s:mv20 |
| 64 | e23 | ребро | d:v8, s:v20 |
| 65 | e24 | ребро | d:v25, s:v27 |
| 66 | e25 | ребро | d:v23, s:v25 |
| 67 | e26 | ребро | d:v22, s:v25 |
| 68 | e27 | ребро | d:v23, s:v23 |
| 69 | e28 | ребро | d:v8, s:m18 |
| 70 | e29 | ребро | d:v8, s:v21 |
| 71 | e30 | ребро | d:v19, s:v21 |
| 72 | e31 | ребро | d:v18, s:v24 |
| 73 | e32 | ребро | d:v8, s:v20 |
| 74 | e33 | ребро | d:mv15, s:mv20 |
| 75 | e34 | ребро | d:v8, s:v10 |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Связи** |
| 76 | e35 | ребро | d:v16, s:mv13 |
| 77 | e36 | ребро | d:v5, s:mv14 |
| 78 | e37 | ребро | d:mv7, s:mv15 |
| 79 | e38 | ребро | d:mv15, s:mv17 |
| 80 | e39 | ребро | d:mv17, s:mv4 |
| 81 | e40 | ребро | d:v12, s:mv5 |
| 82 | e41 | ребро | d:mv13, s:v11 |
| 83 | e42 | ребро | d:v13, s:v14 |
| 84 | e43 | ребро | d:v12, s:v15 |
| 85 | e44 | ребро | d:v9, s:v17 |
| 86 | e45 | ребро | d:v34, s:mv6 |
| 87 | e46 | ребро | d:v34, s:mv10 |
| 88 | e47 | ребро | d:v34, s:v37 |
| 89 | e48 | ребро | d:v35, s:v37 |
| 90 | e49 | ребро | d:v35, s:mv6 |
| 91 | e50 | ребро | d:v15, s:v37 |
| 92 | e51 | ребро | d:v35, s:v36 |
| 93 | e52 | ребро | d:v36, s:v37 |
| 94 | e53 | ребро | d:v38, s:v40 |
| 95 | e54 | ребро | d:v37, s:v39 |
| 96 | e55 | ребро | d:v36, s:v41 |
| 97 | e56 | ребро | d:v39, s:v41 |
| 98 | e57 | ребро | d:v39, s:v40 |
| 99 | e58 | ребро | d:v38, s:v37 |
| 100 | e59 | ребро | d:v34, s:v38 |
| 101 | e60 | ребро | d:mv22, s:mv21 |
| 102 | e61 | ребро | d:mv22, s:v19 |
| 103 | e62 | ребро | d:v41, s:mv22 |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название** | **Тип** | **Связи** |
| 104 | e63 | ребро | d:v7, s:v10 |
| 105 | mv1 | метавершина | ch{v1, v2, v3, v9, e1, e2, e3, e4} |
| 106 | mv2 | метавершина | ch{v2, v3, v6, v7, e3, e6, e10, e5, e13, e63, mv3 } |
| 107 | mv3 | метавершина | ch{v4, v5, v10, e7, e8, e9, e4} |
| 108 | mv4 | метавершина | ch{v11, v12, v13, e15, e40 } |
| 109 | mv5 | метавершина | ch{v13, v14, v15, e42} |
| 110 | mv6 | метавершина | ch{v15, v16, v17} |
| 111 | mv7 | метавершина | ch{v9, v11, v12, e15} |
| 112 | mv8 | метавершина | ch{v5, v10, v8, v21, e36, e8, e34, e29, mv12} |
| 113 | mv9 | метавершина | ch{v30, v7, v29, e11} |
| 114 | mv10 | метавершина | ch{v8, v16, v18, v23, v22, e29, e32, e27, mv15} |
| 115 | mv11 | метавершина | ch{v7, v8, v10, e34, e63} |
| 116 | mv12 | метавершина | ch{v16, e35, mv13} |
| 117 | mv13 | метавершина | ch{mv14} |
| 118 | mv14 | метавершина | ch{v9} |
| 119 | mv15 | метавершина | ch{v19, v21, v20, e30} |
| 120 | mv16 | метавершина | ch{v26, v19, v28, v27, e20, e19, e21, e24, mv18, mv17} |
| 121 | mv17 | метавершина | ch{v22, v23, v25, e27, e25, e26} |
| 122 | mv18 | метавершина | ch{v24, v18, v20, e31} |
| 123 | mv19 | метавершина | ch{v32, v30, v26, e16} |
| 124 | mv20 | метавершина | ch{v31, v32, e13} |
| 125 | mv21 | метавершина | ch{v34, v35, v38, v37, v36, v40, v39, v41, e59, e47, e48, e51, e55, e54, e53, e52, e58, e57, e56} |
| 126 | mv22 | метавершина | ch{v33} |

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Попков В.К. Математические модели связности. Новосибирск: ИВМиМГ СО РАН, 2006. – 490 с.
2. Анохин К.В. Когнитом: гиперсетевая модель мозга // Нейроинформатика-2015. XVII Всероссийская научно-техническая конференция. Сборник научных трудов. Ч. 1. М.: НИЯУ МИФИ. 2015. С. 18.
3. 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.
4. Samokhvalov E.N., Revunkov G.I. Gapanyuk Yu.E. Ispolzovaniye metagraphov 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.
5. 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.
6. 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.
7. Basu A., Blanning R. Metagraphs and their applications. Springer, 2007. 174 p.
8. 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.
9. 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. C. 173-178
10. 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.
11. Wang Q., Mao Z., Wang B., Guo L. Knowledge Graph Embedding: A Survey of Approaches and Applications. IEEE Transactions on Knowledge and Data Engineering, 2017, vol. 29, no. 12, pp. 2724-2743
12. Cai, Hongyun & W. Zheng, Vincent & Chen-Chuan Chang, Kevin. (2017). A Comprehensive Survey of Graph Embedding: Problems, Techniques and Applications. IEEE Transactions on Knowledge and Data Engineering. 10.1109/TKDE.2018.2807452.
13. Дунин И.В., Гапанюк Ю.Е., Ревунков Г.И. Особенности преобразования метаграфа в модель плоского графа. Динамика сложных систем — XXI век, 2018. Выпуск №9. С. 47-51.
14. J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, “Graphx: Graph processing in a distributed dataflow framework,” in OSDI, 2014, pp. 599–613.
15. Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, “Distributed graphlab: A framework for machine learning and data mining in the cloud,” Proc. VLDB Endow., vol. 5, no. 8, pp. 716–727, 2012.
16. P. Kumar and H. H. Huang, “G-store: High-performance graph store for trillion-edge processing,” in SC, 2016, pp. 71:1–71:12.
17. X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, “Community preserving network embedding,” in AAAI, 2017, pp. 203–209.
18. X. Zhao, A. Chang, A. D. Sarma, H. Zheng, and B. Y. Zhao, “On the embeddability of random walk distances,” PVLDB, vol. 6, no. 14, pp. 1690–1701, 2013.
19. X. Zhao, A. Chang, A. D. Sarma, H. Zheng, and B. Y. Zhao, “On the embeddability of random walk distances,” PVLDB, vol. 6, no. 14, pp. 1690–1701, 2013.
20. T. Man, H. Shen, S. Liu, X. Jin, and X. Cheng, “Predict anchor links across social networks via an embedding approach,” in IJCAI, 2016, pp. 1823–1829.
21. T. Pimentel, A. Veloso, and N. Ziviani, “Unsupervised and scal- able algorithm for learning node representations,” in ICLR, 2017.
22. D. Wang, P. Cui, and W. Zhu, “Structural deep network embed- ding,” in KDD, 2016, pp. 1225–1234. [21] S. Cao, W. Lu, and Q. Xu, “Grarep: Learning graph represen- tations with global structural information,” in CIKM, 2015, pp. 891–900.
23. F. Tian, B. Gao, Q. Cui, E. Chen, and T. Liu, “Learning deep representations for graph clustering,” in AAAI, 2014, pp. 1293– 1299. [23] S. Cao, W. Lu, and Q. Xu, “Deep neural networks for learning graph representations,” in AAAI, 2016, pp. 1145–1152.
24. A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, “Distributed large-scale natural graph factoriza- tion,” in WWW, 2013, pp. 37–48.
25. Z. Jin, R. Liu, Q. Li, D. D. Zeng, Y. Zhan, and L. Wang, “Predicting user’s multi-interests with network embedding in health-related topics,” in IJCNN, 2016, pp. 2568–2575.
26. L. Liu, W. K. Cheung, X. Li, and L. Liao, “Aligning users across social networks using network embedding,” in IJCAI, 2016, pp. 1774–1780.
27. C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao, “Scalable graph embedding for asymmetric proximity,” in AAAI, 2017, pp. 2942– 2948.
28. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, “Line: Large-scale information network embedding,” in WWW, 2015, pp. 1067–1077.
29. A. Grover and J. Leskovec, “Node2vec: Scalable feature learning for networks,” in KDD, 2016, pp. 855–864.
30. H. Fang, F. Wu, Z. Zhao, X. Duan, Y. Zhuang, and M. Ester, “Community-based question answering via heterogeneous social network learning,” in AAAI, 2016, pp. 122–128.
31. S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, “Heterogeneous network embedding via deep architec- tures,” in KDD, 2015, pp. 119–128.
32. [34] H. Zhang, X. Shang, H. Luan, M. Wang, and T. Chua, “Learning from collective intelligence: Feature learning using social images and tags,” TOMCCAP, vol. 13, no. 1, pp. 1:1–1:23, 2016.
33. [35] X. Geng, H. Zhang, J. Bian, and T. Chua, “Learning image and user features for recommendation in social networks,” in ICCV, 2015, pp. 4274–4282.
34. F. Wu, X. Lu, J. Song, S. Yan, Z. M. Zhang, Y. Rui, and Y. Zhuang, “Learning of multimodal representations with random walks on the click graph,” IEEE Trans. Image Processing, vol. 25, no. 2, pp. 630–642, 2016.
35. K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, “Freebase: A collaboratively created graph database for structuring human knowledge,” in SIGMOD, 2008, pp. 1247–1250.
36. Maximilian Nickel, Kevin Murphy, Volker Tresp, Evgeniy Gabrilovich, “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: 1.06.2019).
37. F. Wu, J. Song, Y. Yang, X. Li, Z. M. Zhang, and Y. Zhuang, “Structured embedding via pairwise relations and long-range interactions in knowledge base,” in AAAI, 2015, pp. 1663–1670.
38. B. Shi and T. Weninger, “Proje: Embedding projection for knowl- edge graph completion,” in AAAI, 2017, pp. 1236–1242.
39. P. Goyal and E. Ferrara, “Graph embedding techniques, applica- tions, and performance: A survey,” CoRR, vol. abs/1705.02801, 2017.
40. D. Cai, X. He, and J. Han, “Spectral regression: a unified subspace learning framework for content-based image retrieval,” in MM, 2007, pp. 403–412.
41. S. Guattery and G. L. Miller. Graph embeddings and laplacian eigenvalues. SIAM Journal on Matrix Analysis and Applications, 21(3):703–723, 2000.
42. F. R. K. Chung. Spectral Graph Theory, volume 92 of Regional Conference Series in Mathematics. AMS, 1997
43. X. He, S. Yan, Y. Hu, P. Niyogi, and H.-J. Zhang. Face recognition using laplacian faces. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(3):328–340, 2005.
44. T. Hofmann and J. M. Buhmann, “Multidimensional scaling and data clustering,” in NIPS, 1994, pp. 459–466.
45. M. Balasubramanian and E. L. Schwartz, “The isomap algorithm and topological stability,” Science, vol. 295, no. 5552, pp. 7–7, 2002.
46. W. N. A. Jr. and T. D. Morley, “Eigenvalues of the laplacian of a graph,” Linear and Multilinear Algebra, vol. 18, no. 2, pp. 141–145, 1985.
47. Y. Yang, F. Nie, S. Xiang, Y. Zhuang, and W. Wang, “Local and global regressive mapping for manifold learning with out-of- sample extrapolation,” in AAAI, 2010.
48. Y.-Y. Lin, T.-L. Liu, and H.-T. Chen, “Semantic manifold learning for image retrieval,” in MM, 2005, pp. 249–258.
49. R. Jiang, W. Fu, L. Wen, S. Hao, and R. Hong, “Dimensionality reduction on anchorgraph with an efficient locality preserving projection,” Neurocomputing, vol. 187, pp. 109–118, 2016.
50. M. Chen, I. W. Tsang, M. Tan, and C. T. Jen, “A unified feature selection framework for graph embedding on high dimensional data,” IEEE Trans. Knowl. Data Eng., vol. 27, no. 6, pp. 1465–1477, 2015.
51. S. Yan, D. Xu, B. Zhang, H. Zhang, Q. Yang, and S. Lin, “Graph embedding and extensions: A general framework for dimensionality reduction,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 29, no. 1, pp. 40–51, 2007.
52. X. He and P. Niyogi, “Locality preserving projections,” in NIPS, 2003, pp. 153–160.
53. S. T. Roweis and L. K. Saul, “Nonlinear Dimensionality Reduction by Locally Linear Embedding,” Science, vol. 290, no. 5500, pp. 2323–2326, 2000.
54. Y. Yang, F. Nie, S. Xiang, Y. Zhuang, and W. Wang, “Local and global regressive mapping for manifold learning without-of sample extrapolation,” in AAAI, 2010.
55. S. Xiang, F. Nie, C. Zhang, and C. Zhang, “Nonlinear dimensionality reduction with local spline embedding,” IEEE Trans. Knowl. Data Eng., vol. 21, no. 9, pp. 1285–1298, 2009.
56. L. Vandenberghe and S. Boyd, “Semidefinite programming,” SIAM Rev., vol. 38, no. 1, pp. 49–95, 1996.
57. K. Q. Weinberger, F. Sha, and L. K. Saul, “Learning a kernel matrix for nonlinear dimensionality reduction,” in ICML, 2004.
58. T. M. V. Le and H. W. Lauw, “Probabilistic latent document network embedding,” in ICDM, 2014, pp. 270–279.
59. G. H. Golub and C. Reinsch, “Singular value decomposition and least squares solutions,” Numer. Math., vol. 14, no. 5, pp. 403–420, 1970.
60. L. Yao, Y. Zhang, B. Wei, Z. Jin, R. Zhang, Y. Zhang, and Q. Chen, “Incorporating knowledge graph embeddings into topic modeling,” in AAAI, 2017, pp. 3119–3126.
61. M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, “Asymmetric transitivity preserving graph embedding,” in KDD, 2016, pp. 1105–1114.
62. A. Ahmed, N. Shervashidze, S. Narayanamurthy, V. Josifovski, and A. J. Smola, “Distributed large-scale natural graph factorization,” in WWW, 2013, pp. 37–48.
63. C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, “Network representation learning with rich text information,” in IJCAI, 2015, pp. 2111–2117.
64. C. Tu, W. Zhang, Z. Liu, and M. Sun, “Max-margin deepwalk: Discriminative learning of network representation,” in IJCAI, 2016, pp. 3889–3895.
65. Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’14, pages 701–710, New York, NY, USA, 2014. ACM.
66. Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.
67. А. Б. Теслюк. О случайном блуждании по графу веб-документов. – М.: Московский физикотехнический институт, 2004. – С. 4-5.
68. Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.
69. Aditya Grover, Jure Leskovec. node2vec, Scalable feature learning for networks. Phoenix, Arizona: International Conference on Knowledge Discovery and Data Mining, 2016. – 4 p.
70. X. Wang, P. Cui, J. Wang, J. Pei, W. Zhu, and S. Yang, “Community preserving network embedding,” in AAAI, 2017, pp. 203–209.
71. F. Nie, W. Zhu, and X. Li, “Unsupervised large graph embedding,” in AAAI, 2017, pp. 2422–2428.
72. C. Zhou, Y. Liu, X. Liu, Z. Liu, and J. Gao, “Scalable graph embedding for asymmetric proximity,” in AAAI, 2017, pp. 2942– 2948.
73. X. Wei, L. Xu, B. Cao, and P. S. Yu, “Cross view link prediction by learning noise-resilient representation consensus,” in WWW, 2017, pp. 1611–1619.
74. G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. Science, 2006.
75. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.
76. D. Wang, P. Cui, M. Ou, and W. Zhu. Deep multimodal hashing with orthogonal regularization. In Proceedings of the 24th International Conference on Artificial Intelligence, pages 2291–2297. AAAI Press, 2015.
77. K. Georgiev and P. Nakov. A non-iid framework for collaborative filtering with restricted boltzmann machines. In ICML-13, pages 1148–1156, 2013.
78. F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu. Learning deep representations for graph clustering. In Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, pages 1293–1299, 2014.
79. ] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang. Heterogeneous network embedding via deep architectures. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 119–128. ACM, 2015.
80. Воронцов К.В., 2010. Лекции по алгоритмам кластеризации и многомерного шкалирования. URL: http://www.ccas.ru/voron/download/Clustering.pdf (дата обращения: 16 июня 2019)
81. F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu. Learning deep representations for graph clustering. In AAAI, 2014.
82. API Reference — scikit-learn 0.21.2 documentation [Электронный ресурс] // URL: http://academim.org/art/pan1\_2.html (дата обращения: 17.06.2019).
83. 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
84. D. Zhang, J. Yin, X. Zhu and C. Zhang, “MetaGraph2Vec: Complex Semantic Path Augmented Heterogeneous Network Embedding”, PAKDD, pp. 196-208, 2018.
85. ZHANG, Wentao; FANG, Yuan; LIU, Zemin; WU, Min; and ZHANG, Xinming. mg2vec: Learning relationship-preserving heterogeneous graph representations via metagraph embedding. (2020). IEEE Transactions on Knowledge and Data Engineering. 1-14. Research Collection School Of Information Systems.