О возможности совместного использования метаграфовых и миварных моделей.

# Сложные сети

В настоящее время все большее применение находят модели на основе сложных сетей. С их помощью решаются задачи, как из технических, так и из естественных наук. Существует много моделей, описывающих сложные сети, среди таких моделей на нашей кафедре широкое распространение получили метаграфовая и миварная модель.

# Метаграфовые модели

Метаграфовая модель представляет собой модель сложного графа с эмерджентностью, которая реализуется с помощью метавершин и метаребер. (надо добавить что-то еще)

# Миварные модели

Миварный подход основывается на двух основных технологиях: миварная технология накопления знаний и миварная технология обработки информации.

1) *Миварная технология накопления информации* - это способ создания глобальных эволюционных баз данных и правил (знаний) с изменяемой структурой на основе адаптивного дискретного миварного информационного пространства унифицированного представления данных и правил, базирующегося на трех основных понятиях "вещь, свойство, отношение".

2) *Миварная технология обработки информации* - это способ создания системы логического вывода или "автоматического конструирования алгоритмов из модулей, сервисов или процедур" на основе активной обучаемой миварной сети правил с линейной вычислительной сложностью.

Миварная технология способна осуществлять логический вывод, поддерживает вычислительные процедуры и «сервисы».

Важной особенностью технологии миварного вывода является то, что, в отличие от нейронных сетей, она работает по принципу «белый ящик». То есть оператор системы может проконтролировать этапы принятия решения и скорректировать их в случае необходимости.

Таким образом, миварные сети позволяют развить продукционный подход и создать логически рассуждающую систему. При этом в [2] утверждается, что логический вывод происходит с линейной сложностью.

# Цели объединения

Как видно из написанного выше, метаграфовые модели являются удобным средством представления знаний. Однако, ввиду отсутствия отлаженного средства обработки информации, логический вывод на метаграфе является затруднительным.

Данного недостатка решена миварная модель. Она обладает отлаженным «движком», на котором успешно функционирует большое количество интеллектуальных систем [1]. В свою очередь миварное представление знаний в виде набора правил является сложным для человеческого восприятия.

Объединение метаграфовой и миварной моделей позволило бы взять лучшее от каждой из них и скомпенсировать их недостатки. Таким образом, авторами статьи предлагается разработать гибридную систему, в которой будет использоваться метаграфовая в качестве технологии накопления информации и миварная технология обработки информации.

# Допущения, ограничения для метаграфовой модели

Основные положения метаграфовой модели приведены в пособии [3].

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

**Вершина метаграфа** характеризуется множеством атрибутов:

vi = {atrk}, viV

где *vi* – вершина метаграфа; *atrk* – атрибут.

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

где *ei* – ребро метаграфа; *vS* – исходная вершина (метавершина) ребра; *vE* – конечная вершина (метавершина) ребра; *eo* – признак направленности ребра (*eo=true* – направленное ребро, *eo=false* – ненаправленное ребро); *atrk* – атрибут.

**Фрагмент метаграфа:**

где *MGi* – фрагмент метаграфа; *evj* – элемент, принадлежащий объединению множеств вершин (метавершин) и ребер (метаребер) метаграфа.

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

**Метавершина метаграфа:**

где *mvi* – вершина метаграфа; *atrk* – атрибут, *evj* – элемент, принадлежащий объединению множеств вершин (метавершин) и ребер (метаребер) метаграфа.

Таким образом, метавершина в дополнение к свойствам вершины включает вложенный фрагмент метаграфа. При этом ребра и метаребра этого фрагмента могут быть только ненаправленными, *eo=false*.

**Метаребро метаграфа:**

где *mei* – метаребро метаграфа; *vS* – исходная вершина (метавершина) ребра; *vE* – конечная вершина (метавершина) ребра; *eo* – признак направленности метаребра (*eo=true* – направленное метаребро, *eo=false* – ненаправленное метаребро); *atrk* – атрибут; *evj* – элемент, принадлежащий объединению множеств вершин (метавершин) и ребер (метаребер) метаграфа.

Для того чтобы появилась возможность транслировать метаграфовое представление в миварную сеть, необходимо внести в данные правила небольшие допущения, которые в основном касаются ребер метаграфа:

1. В формальном описании метаграфа ребро может быть как направленным, так и ненаправленным. Так как нашей целью является транслирование метаграфовой модели в миварные правила, которые можно описать конструкцией «Если…, то…», то данное правило является для нас неудобным. Предположим, что все ребра в нашей объединенной модели являются направленными. Тогда ненаправленная связь может быть смоделирована с помощью двух ребер, начальная вершина одной из которых совпадает с конечной вершиной другой и наоборот.



1. В формальном описании модели метаграфа ребро характеризуется одной начальной вершиной и одной конечной. Несложно представить ситуацию, в которой одной связью может быть связано более двух объектов (вершин). Поэтому, мы будем характеризовать ребро множеством начальных вершин и множеством конечных вершин. (надо придумать как это красиво представить графически)

# Допущения, ограничения миварной модели

Миварный подход позволяет работать с различными формами представления данных и правил (знаний), включая и работу с бесконечными описаниями сущностей - вещей, отношений и свойств в миварном многомерном динамическом информационном пространстве унифицированного представления данных и правил (знаний).

Для трансляции метаграфа в мивар нам потребуется возможность мивара работать с:

1. Переменными сложных типов;
2. Связями, которые доопределяются во время выполнения программы.

Первое требование поддерживается миварной системой вывода, поэтому с ним не возникает никаких проблем. Второе же условие может быть реализовано с помощью добавления к функции, представляющей связь дополнительных переменных, которые будут относиться к множеству входных переменных и представлять собой внутреннее состояние связи.

# Переход от метаграфовой модели к миварной

Рассмотрим, как будут преобразовываться основные элементы модели метаграфа при переходе из метаграфовой модели к миварной.

**Атрибут** является конструкцией типа «Ключ-Значение». Атрибут не может быть самостоятельным элементом метаграфовой модели, однако он является основным строительным материалом для всех остальных элементов метаграфовой модели. В миварной модели атрибут может быть представлен переменной(столбцом) простого типа.

Миварная модель логического вывода способна работать как с переменными простых, так и с переменными сложных типов. **Вершина** метаграфа в зависимости от количества атрибутов в ней может быть представлена переменной (столбцом) простого или сложного типа, состоящей из переменных простых типов.

**Связи** метаграфа в миварном представлении могут выглядеть как функции (строки). При этом они могут представлять собой функции на каком-либо языке программирования или являться отдельными программами. Соответственно связи могут соединять вершины и метавершины, что соответствует и метаграфовой, и миварной моделям.

**Метавершина** представляет собой набор других вершин, метавершин, связей и атрибутов. В миварном представлении она может быть представлена как переменная (столбец) сложного типа, внутренние элементы которой могут быть связаны посредством связей. Чтобы сохранит свойство модели передавать эмерджентность, при переходе от метаграфовой модели к миварной для каждой метавершины необходимо добавить специальную связь(строка)-конструктор, которая будет определять новые свойства, образующиеся в системе. Данная связь имеет много общего с функцией-конструктором, применяемой в объектно-ориентированных языках. Также, так как связь метавершины с составляющими ее элементами по сути является ненаправленной, нам может понадобиться обратная связь-конструктор, которая будет принимать на вход метавершину, и определять значения вершин и атрибутов этой метавершины.

**Метасвязь** отличается от связи тем, что зависит от своего внутреннего состояния. Как было упомянуто выше, данное поведение может быть промоделировано путем добавления к множеству входных вершин связи дополнительных вершин (метавершин), которые являются внутренними вершинами метасвязи. При этом между внутренними связями метавершины могут существовать свои связи.

Дальше рассмотрим два примера трансляции метаграфа в мивар.

# Пример с эмерджентностью (двухкомпонентный клей)

Приведем пример, демонстрирующий преобразование метавершины из метаграфовой модели в миварную.

Предположим, что у нас есть двухкомпонентный клей, состоящий из двух компонентов: A и B.

Агрегатное состояние каждой компоненты клея по отдельности – жидкость. Однако при смешивании мы получаем химическую реакцию, которая приводит к тому, что агрегатное состояние их смеси – твердое тело.

Данный пример может быть представлен в виде метаграфа следующим образом.



На данной схеме овалами изображены атрибуты, прямоугольниками – вершины.

Связь-конструктор, связывающая двухкомпонентный клей с его компонентами, должна проверить химические вещества, входящие в компоненты клея и, если они являются правильными, задать значение атрибута клея «Агрегатное состояние» как «Твердое тело». Входными вершинами данной связи будут являться компоненты А и B. Выходная вершина связи конструктора – Двухкомпонентный клей.

Если использовать синтаксис языка Python3, функция конструктор может быть записана в следующем виде:

class Двухкомпонентный\_клей:

def \_\_init\_\_( self, компонента\_А, компонента\_B ):

if( компонента\_А.химическое\_вещество == "Дифенилолпропан" and \

компонента\_B.химическое\_вещество == "Эпихлоргидрин"):

self.агрегатное\_состояние = "Твердое тело"

else:

self.агрегатное\_состояние = "Жидкость"

def обратный\_конструктор(self, компонента\_А, компонента\_B, агрегатное\_состояние ):

компонента\_А = self.компонента\_А

компонента\_B = self.компонента\_B

агрегатное\_состояние = self.агрегатное\_состояние

В случае если возникнет необходимость определить, в каком агрегатном состоянии будет находиться двухкомпонентный клей, миварная система вывода получит на вход следующую матрицу:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Связи | Компонента\_А | Компонента\_B | Компонента\_А.Агрегатное\_состояниие | Компонента\_B.Агрегатное\_состояние | Компонента\_А.Химическое\_вещество | Компонента\_B.Химическое\_вещество | Двухкомпонентный\_клей | Двухкомпонентный\_клей.Агрегатное\_состояние |
| Двухкомпонентный\_клей.\_\_init\_\_ | x | x |  |  |  |  | y |  |
| Компонента\_А.\_\_init\_\_ | y |  | x |  | x |  |  |  |
| Компонента\_B.\_\_init\_\_ |  | y |  | x |  | x |  |  |
| Двухкомпонентный клей.обратный\_конструктор | y | y |  |  |  |  | x | y |
| Служебная информация |  |  | z | z | z | z |  | w |

Здесь буквой x помечаются входные вершины связи, буквой y – выходные вершины, буквой z – известные вершины и буквой w – искомые.

После применения алгоритма миварного вывода, получим последовательность функций, выполнение которой приведет нас к результату.

Этот пример демонстрирует то, как проявление эмерджентности может быть перенесено из метаграфовой модели в миварную.

# Пример решения задачи (задача по физике)

Рассмотрим функционирование объединенной метаграфово-миварной системы на примере простой школьной задачки по физике.

«Пуля массой 9г, летевшая со скоростью 600м/с, попадает в деревянную стену и проникает в нее на глубину 20 см. Определить среднюю силу сопротивления движению пули.»

Метаграф для данной задачи может быть представлен следующим образом:



В текстовом представлении, описанном в [4]:

Metavertex(Name=Bullet,

Vertex(Name:=mass, Value:=0.009),

Vertex(Name:=speed, Value:=600),

Vertex(Name:=energy),

Edge(Name:=e1, V1:=speed, V2:=mass, V3:=energy), \\* здесь и далее V1 – выходная вершина, V2 и V3 – входные \*/

Edge(Name:=e2, V1:=mass, V2:=speed, V3:=energy),

Edge(Name:=e3, V1:=energy, V2:=speed, V3:=mass)

)

Metavertex(Name=Wall,

Vertex(Name:=energy,

Vertex(Name:=depth, Value:=0.02),

Vertex(Name:=resistance\_force),

Edge(Name:=e4, V1:= energy, V2:= depth, V3:= resistance\_force),

Edge(Name:=e4, V1:= depth, V2:= energy, V3:= resistance\_force),

Edge(Name:=e4, V1:= resistance\_force, V2:= energy, V3:= depth)

)

Metaedge(Name=to\_slow\_down

Vertex(Name:=end\_speed, Value:=0)

)

Metaedge(Name=to\_penetrate

Vertex(Name:=end\_speed, Value:=0)

)

Связи могут быть функциями на любом языке или целыми программами, реализация которых хранится отдельно. Однако для нашего примера мы будем использовать язык Python.

def e1(mass, energy):

return math.sqrt( 2 \* energy / mass )

def e2(speed, energy):

return 2 \* energy / ( speed \*\* 2 )

def e3(speed, mass):

return mass \* ( speed \*\* 2 ) / 2

class Bullet:

def \_\_init\_\_( self, mass, energy, speed );

self.mass=mass

self.energy=energy

self.speed=speed

def reverse\_constructor( self, mass, energy, speed ):

if( mass == NULL ):

mass = self.mass

if( energy == NULL ):

energy = self.energy

if( speed == NULL ):

speed = self.speed

def e4 (depth, resistance\_force):

return depth \* resistance\_force

def e5(energy, resistance\_force):

return energe / resistance\_force

def e6(energy, depth):

return energy / depth

class Wall:

def \_\_init\_\_( self, depth, energy, resistance\_force):

self.depth = depth

self.resistance\_force=resistance\_force

self.energy=energy

def reverse\_constructor( self, depth, energy, resistance\_force):

if( depth == NULL ):

depth = self.depth

if( energy == NULL ):

energy == self.energy

if( resistance\_force == NULL ):

resistance\_force = self.resistance\_force

def to\_slow\_down(bullet, wall, end\_speed):

wall.energy = bullet.energy - bullet.mass \* (end\_speed \*\* 2) / 2

def to\_penetrate(bullet, wall, end\_speed):

bullet.energy = wall.energy + bullet.mass \* (end\_speed \*\* 2) / 2

Теперь можем применить миварный вывод.

В начальный момент имеем матрицу следующего вида:

# Возможная архитектура системы

На основе рассмотренного в статье метода возможно создание модуля логического вывода, который будет являться модулем сознания архитектуре ГИИС. В самом простом случае данный модуль может использоваться с двумя модулями подсознания, осуществляющими конвертацию естественного языка в атрибуты метаграфа и атрибуты метаграфа в естественный язык.



Данная система будет представлять из себя слабый искусственный интеллект в предметной области логического модуля. Она будет способна вести диалог с пользователем и логически решать его задачи.

# Выводы

# Литература

1. Варламов О.О. О СОЗДАНИИ МИВАРНЫХ ЭКСПЕРТНЫХ СИСТЕМ НА ОСНОВЕ «МНОГОМЕРНОЙ ОТКРЫТОЙ ГНОСЕОЛОГИЧЕСКОЙ АКТИВНОЙ СЕТИ» MOGAN. ОБЗОР ПРАКТИЧЕСКИХ ПРИМЕРОВ-2020 УДК 004.8 + 007.5
2. Варламов О.О. ПЕРЕХОД ОТ ПРОДУКЦИЙ К ДВУДОЛЬНЫМ МИВАРНЫМ СЕТЯМ И ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ АВТОМАТИЧЕСКОГО КОНСТРУКТОРА АЛГОРИТМОВ, УПРАВЛЯЕМОГО ПОТОКОМ ВХОДНЫХ ДАННЫХ И ОБРАБАТЫВАЮЩЕГО БОЛЕЕ ТРЕХ МИЛЛИОНОВ ПРАВИЛ УДК 004.82+007.52
3. Гапанюк Ю.Е. Конспект лекций по спецкурсу «Гибридные интеллектуальные информационные системы на основе метаграфового подхода»: Учебно-методическое пособие. – М.: Издательство «Спутник +», 2018. – 53с., ил. УДК 004.8
4. Гапанюк Ю.Е. Подход к разработке метаграфового исчисления УДК 004.6