УДК 004.82
DOI:10.38028/ESI.2022.28.4.017

Метод поиска и логического вывода экспертной информации в ориентированном циклическом мультиграфе знаний

Зайцев Анатолий Федорович
Восточно-Сибирский государственный университет технологий и управления,
Россия, Улан-Удэ, *lordsadler2010@mail.ru*

Аннотация. В настоящее время многие современные задачи прикладной математики и информатики решаются с применением теории графов. В виде графов можно представлять и описывать различные сложные системы, например, нейросети или базы знаний. Наиболее часто встречающимися задачами с применением теории графов являются: поиск кратчайшего пути, определение максимального потока в сети, поиск минимальных остовных деревьев и другие. В то же время, существует достаточно много нерешенных проблем. Актуальность работы вызвана повышением интереса к областям искусственного интеллекта и инженерии знаний, методы которых заключаются в возможности трансформации полученных предметных моделей в логико-математические, в виде компьютерных программ, осуществляющих компьютерное или имитационные моделирование исследуемых систем. В представленной работе описан процесс разработки специального математического и алгоритмического обеспечения для систем анализа и обработки экспертной информации с целью автоматизированного поиска, и синтеза решений задач моделирования и идентификации динамических систем. Сформулирована проблема поиска и логического вывода синтезированных решений задач на графах знаний. Представлена модель базы знаний выбранной предметной области в виде мультиграфа знаний, а также новый модифицированный метод поиска и логического вывода решений задач, с их программной реализацией на языке программирования Python. Представленный метод поиска слабосвязанных подграфов и синтеза решений задач реализован с применением теоретико-множественного анализа, а также элементов теории графов. Для демонстрации работоспособности метода приводится его реализация в виде алгоритма на языке программирования Python и результаты вычислительных экспериментов. Новизна и практическая значимость работы заключаются в том, что предложенный метод и алгоритмы могут быть использованы при практической реализации баз знаний, механизмов логического вывода и обработки экспертной информации в различного рода экспертных, расчетно-логических и гибридных интеллектуальных системах, замещающих собой зарубежные аналоги.

Ключевые слова: системный анализ, логический вывод, поиск на графах, мультиграф, экспертная система, база знаний, граф знаний, Python

Цитирование: Зайцев А.Ф. Метод поиска и логического вывода экспертной информации в ориентированном циклическом мультиграфе знаний / А.Ф. Зайцев // Информационные и математические технологии в науке и управлении. – 2022. – № 4(28). – С. 213-222. – DOI:10.38028/ESI.2022.28.4.017.

Введение. В представленной работе описан процесс разработки специального математического и алгоритмического обеспечения для систем анализа и обработки информации с целью автоматизированного поиска и синтеза решений задач моделирования и идентификации динамических систем. Подобные экспертные системы должны содержать базу знаний о некоторой предметной области, а также метод логического вывода и обработки экспертной информации.

В качестве предметной области для базы знаний выбрана область механики, в рамках которой может быть выполнено решение множества различных задач. В качестве математической основы для базы знаний используется описание понятий и величин из области механики, представленное в виде формальной грамматики [1, 2]. Базу знаний в определенный момент можно пополнять, внося новые правила и функции. Чем больше правил содержится в базе, тем больший круг задач она способна решить в рамках выбранной предметной области. В представленной работе база знаний трансформируется и представляется в виде ориентированного, невзвешенного, циклического мультиграфа знаний. Графы знаний отличаются от классических графов тем, что они, как правило представляют собой направленные мультиграфы с типизированными ребрами. Подробнее о графах знаний можно узнать в работах [3-6].

В работе рассматривается проблема поиска и логического вывода синтезированных решений задач на графе знаний. Известно, что для задач поиска остовных деревьев для неориентированных взвешенных графов существуют алгоритмы Прима, Крускала и Борувки [7, 8]. Также существует задача нахождения строго минимального остовного дерева Штейнера [9, 10]. Для нахождения слабосвязанных подграфов в неориентированных взвешенных графах, с положительным или отрицательным весом ребер применяется алгоритм Флойда–Уоршелла и его модификации [7]. Данные методы и их алгоритмы не подходят для решения рассматриваемой проблемы, так как её отличие заключается в ориентированности, невзвешенности и цикличности исходного мультиграфа знаний, а также в том, что правильный найденный подграф (или дерево), представляющий собой решение, может быть как строго, так и нестрого минимальным. Проблема поиска и логического вывода синтезированных решений задач на графах знаний – является NP-трудной.

В представленной работе предложен новый модифицированный метод поиска и логического вывода синтезированных решений задач на графе знаний, с его программной реализацией на языке программирования Python. Предложенный метод решения рассматриваемой проблемы и его алгоритм могут быть использованы при реализации механизмов логического вывода в различных экспертных, расчетно-логических и гибридных интеллектуальных системах.

1. Цель исследования. Целью исследования является решение проблемы поиска и логического вывода синтезированных решений задач в ориентированном, невзвешенном, циклическом мультиграфе знаний. Полученные решения должны представлять собой ациклические слабосвязанные подграфы (или деревья), состоящие из определенных подмножеств указанных известных (терминальных) вершин, а также некоторых неизвестных (нетерминальных) вершин, соединяющих между собой известные.

2. Материалы и методы исследования. В качестве материалов исследования используются: компьютеры и инструментальные средства программного обеспечения, язык программирования Python.

В процессе исследования задействованы следующие методы: анализ, синтез, формализация, моделирование, алгоритмизация, вычислительный эксперимент, элементы теории множеств и теории графов.

3. Формальное описание базы знаний. Имеется исходная формальная модель базы знаний (KB), описанная следующей грамматикой:

$$KB = \{Vt, Vn, P, S, F\},$$

где Vt – множество терминальных символов, обозначающих изначально известные величины, используемые в процессе поиска решения какой-либо задачи; Vn – множество нетерминальных символов, обозначающих неизвестные величины в решаемой задаче; P – множество правил вывода:

$\{$ $Spd2 \rightarrow Spd1, Accl, Time$; $\quad$ $Spd2 \rightarrow Accl, Dist, Spd1$;
$Spd2 \rightarrow KinEn2, Mass$; $\quad$ $Spd1 \rightarrow Spd2, Accl, Time$;
$Spd1 \rightarrow Dist1, Time, Accl$; $\quad$ $Spd1 \rightarrow Spd2, Accl, Dist$;
$Spd1 \rightarrow KinEn1, Mass$; $\quad$ $Dist \rightarrow Spd1, Time, Accl, Time$;
$Dist \rightarrow Spd2, Spd1, Accl$; $\quad$ $Dist \rightarrow OpPull, FrPull$;
$Dist \rightarrow OpFric, FrFric$; $\quad$ $Time \rightarrow Spd2, Spd1, Accl$;
$Time \rightarrow Spd1, Accl, Dist$; $\quad$ $Accl \rightarrow Spd2, Spd1, Time$;
$Accl \rightarrow Dist, Spd1, Time$; $\quad$ $Accl \rightarrow Spd2, Spd1, Dist$;
$Accl \rightarrow FrPull, FrFric, Mass$; $\quad$ $Mass \rightarrow FrPull, FrFric, Accl$;
$Mass \rightarrow FrFric, CfFric, Accl$; $\quad$ $Mass \rightarrow KinEn2, Spd2$;
$Mass \rightarrow KinEn1, Spd1$; $\quad$ $FrPull \rightarrow FrFric, Mass, Accl$;

$FrPull \to OpPill, Dist$; $FrFric \to FrPull, Mass, Accl$;
$FrFric \to Cfg, CfFric, Mass$; $FrFric \to OpFric, Dist$;
$CfFric \to FrFric, Mass, Cfg$; $KinEn2 \to Mass, Spd2$;
$KinEn2 \to OpPull, OpFric, KinEn1$; $KinEn1 \to Mass, Spd1$;
$KinEn1 \to KinEn2, OpPull, OpFric$; $OpPull \to FrPull, Dist$;
$OpPull \to KinEn2, KinEn1 OpFric$; $OpFric \to FrFric, Dist$;
$OpFric \to OpPull, KinEn2, KinEn1$; };

S – нетерминальный символ из множества Vn, означающий то, что нужно найти в процессе решения; F – множество функций, ассоциативно связанных с подмножествами правил вывода P, необходимых для синтеза математических выражений и вывода решений задач. Множество правил вывода P фактически включает в себя полный алфавит всех используемых символов.

Обозначения символов алфавита грамматики следующие: «*Dist*» – расстояние; «*Time*» – время; «*Spdl*» – начальная скорость; «*Spd2*» – конечная скорость; «*Accl*» – ускорение; «*Mass*» – масса; «*kinEnl*» – начальная кинетическая энергия; «*kinEn2*» – конечная кинетическая энергия; «*FrPull*» – сила тяги; «*FrFric*» – сила трения; «*OpPull*» – работа силы тяги; «*OpFric*» – работа силы трения; «*CfFric*» – коэффициент трения; «*Cfg*» – ускорение свободного падения.

Так как правила грамматики имеют строго заданную направленную последовательность вывода, то множество правил вывода P можно преобразовать в направленный (ориентированный) мультиграф, представив его в виде списков смежности. В таком случае, например, правило «*Spd2 → Spd1, Accl, Time*» будет означать то, что из вершины «*Spd2*» исходят связанные с ней вершины: «*Spd1*», «*Accl*» и «*Time*».

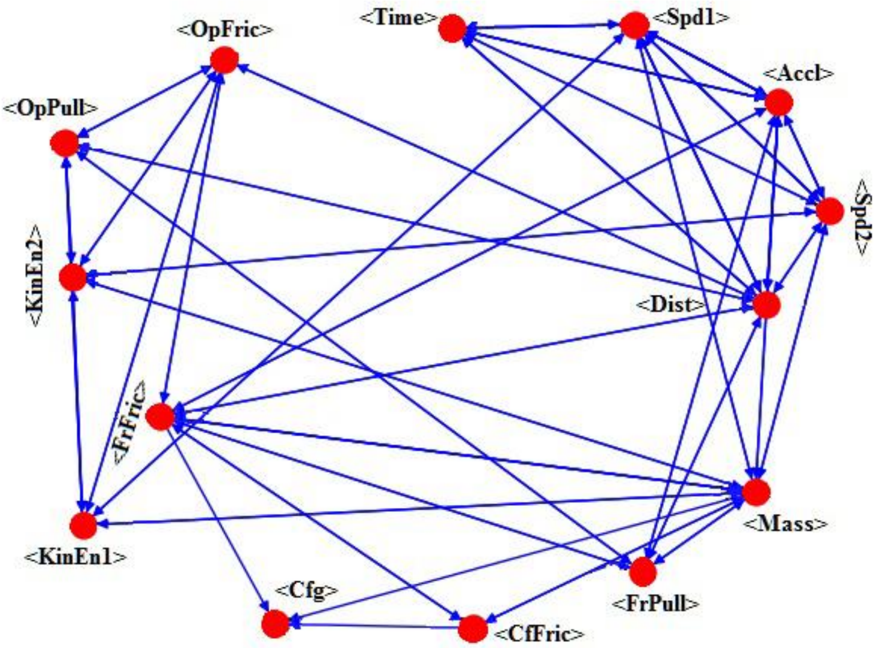

Рис. 1. Ориентированный циклический мультиграф знаний

Используя данный принцип, после преобразования всех правил получим мультиграф знаний $G(V, E)$, в котором V – множество всех вершин, символически описывающих понятия и величины выбранной предметной области, а E – множество связей между вершинами и подмножествами других вершин, сформированных на основе преобразования из правил вывода P. Мультиграф знаний $G(V, E)$ изображен на рисунке 1.

Модифицированную формальную модель базы знаний (*KB*), можно представить следующем образом:

$$KB = \{Vt, G(V, E), S, F\}$$

Теперь, в мультиграфе знаний $G(V,E)$, описывающем понятия из области механики, можно будет выполнять поиск и синтез решений задач. Рассмотрим вышеизложенное на примере решения конкретной задачи. Пусть в решаемой задаче дано: расстояние «*Dist*», начальная кинетическая энергия «*KinEn1*», конечная кинетическая энергия «*KinEn2*», сила тяги «*FrPull*». При этом требуется найти силу сопротивления «*FrFric*». Из данной постановки задачи становятся известны пять величин, которые являются некоторым подмножеством множества всех вершин V, мультиграфа G.

Известно, что для решения представленной в качестве примера задачи сначала необходимо найти работу силы тяги «*OpPull*», умножив силу тяги «*FrPull*» на расстояние «*Dist*», а затем вычислить работу силы сопротивления «*OpFric*» равную: «*OpPull*» – «*KinEn2*» + «*KinEn1*». После этих действий можно будет найти силу сопротивления «*FrFric*», разделив работу силы сопротивления «*OpFric*» на расстояние «*Dist*». Из этого следует, что необходимый подграф, кроме заданных пяти величин, должен включать в себя еще и две ранее неизвестные величины: силу тяги «*OpPull*» и силу сопротивления «*OpFric*».

Формально, синтез решения данной задачи будет представлять собой поиск в мультиграфе знаний $G(V,E)$ такого ациклического слабосвязанного подграфа, который должен содержать в себе подмножество указанных известных (терминальных) вершин, а также подмножество некоторых неизвестных (нетерминальных) вершин, соединяющих между собой известные.

Если такой подграф будет найден, то это будет означать, что задача имеет решение. В противном случае задача будет считаться не имеющей решения. В итоге, найденный слабосвязный подграф, представляющий собой решение задачи, будет иметь вид (рисунок 2).

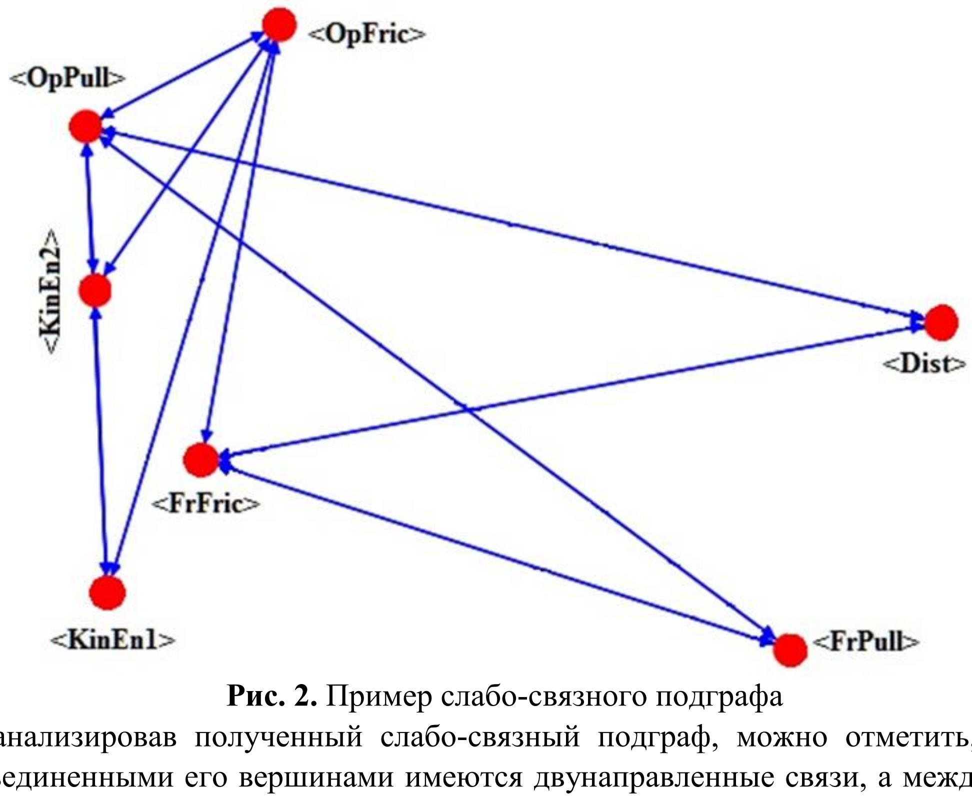

Рис. 2. Пример слабо-связного подграфа

Проанализировав полученный слабо-связный подграф, можно отметить, что между всеми объединенными его вершинами имеются двунаправленные связи, а между найденными ранее неизвестными (нетерминальными) вершинами существует некоторая последовательность (путь).

Таким образом, можно предположить, что синтез правильного решения рассматриваемой задачи будет представлять собой строгую последовательность (путь), состоящую из нетерминальных вершин, которая обязательно должна начинаться с вершины, обозначенной в задаче как то, что требуется найти. В приведенном примере такой вершиной является

«*FrFric*» и правильной последовательностью будет считаться: «*FrFric*» → «*OpFric*» → «*OpPull*».

4. Формальное описание метода. Имеется ориентированный, невзвешенный, циклический мультиграф знаний $G(V, E)$, представленный в виде списков смежности. Предлагается следующий метод поиска в нем нужного слабо-связного подграфа, представляющего собой решение некоторой задачи.

На первом этапе необходимо избавиться от некоторого количества циклов в исходном графе. Для этого необходимо удалить из графа однонаправленные исходящие связи от терминальных вершин Vt, которые становятся известны из постановки задачи. Это можно сделать путём удаления терминальных вершин Vt из множества всех вершин V графа $G(V, E)$, используя операцию разности множеств:

$$G = \{\, V = (V \backslash Vt), E \,\} \tag{1}$$

На следующем этапе для каждой вершины, начиная с вершины-аксиомы, выполняется циклический обход графа, в процессе которого осуществляется проверка наличия связей всех найденных нетерминальных вершин с терминальными. На каждой итерации обхода будет доступен постоянно пополняющийся список всех посещенных терминальных и нетерминальных вершин *VR*, а также осуществляются дополнительные проверки на присутствие в нем вершины-аксиомы, чтобы избежать бесконечно повторяющегося обхода.

$$VR = \Big\{ \big\{ U_{i=1}^{n} G(V, E)_i \,, \{G(E)_i\}, \{S\}, \{i\} \big\} \,|\, G(V)_i = S \Big\}, \tag{2}$$

где i – индекс выбранной вершины с подмножествами ее связей, n – кол-во всех вершин в графе.

После первого обхода и заполнения списка *VR*, циклический обход повторяется еще n раз, но уже по списку *VR*. На данном этапе вновь проверяются связи всех найденных подмножеств нетерминальных вершин с терминальными и пропускаются ранее посещенные вершины.

$$VR = \left\{ VR_{[i][1]} = \cup_{j=1}^{n} (Item_j = \cup_{i=1}^{n} VR_{[i][1]})_j \,|\, Item_j \notin Vt, Item_j \in G(V), Item_j \notin VR_{[i][2]} \right\},$$

$$VR = \left\{ \cup_{k=1}^{n} G(\, \cup_{j=1}^{n} (Item_j = \cup_{i=1}^{n} VR_{[i][1]})_k, E), \{G(E)_k\}, \left\{Item_j\right\}, \{j\} \,|\, G(V)_k = Item_j \right\}, \tag{3}$$

где i, j, k – индексы выбранных связанных вершин, $Item_j$ – j-я выбранная вершина из подмножеств связей, удовлетворяющая заданным условиям, n – кол-во всех вершин.

На последнем этапе происходит операция вычисления разницы между множеством известных терминальных вершин Vt и множеством элементов из списка всех посещенных терминальных и нетерминальных вершин *VR*. Если все терминальные вершины будут обнаружены в списке посещенных вершин, то осуществляется переход к выводу правильной последовательности (пути) из нетерминальных вершин и синтеза решения задачи.

Стоит отметить, что обход графа может быть выполнен не полностью и может завершиться раньше, как только будут найдены связи обнаруженных нетерминальных вершин со всеми терминальными вершинами. Обход графа завершается успешно при условии, если были найдены связи со всеми терминальными вершинами. Если в процессе полного обхода не будут найдены связи со всеми терминальными вершинами, то процесс поиска завершается, а задача помечается как не имеющая решения.

5. Реализация метода в виде алгоритма. Для реализации алгоритма был выбран язык программирования Python [11]. Данный язык довольно популярен в настоящее время и часто используется для разработки различных алгоритмов, за счет простого и краткого синтаксиса, а также наличия всех необходимых структур обработки данных. Для представления графа в виде списков смежности в языке Python была использована структура данных «OrderedDict» из стандартного модуля «collections».

Структура «OrderedDict» представляет собой упорядоченный словарь множества пар, вида: «ключ ↦ значение». В качестве ключа в такой структуре можно задавать вершины графа, а в качестве значений – связанные с ними множества вершин. Подобные структуры также называют ассоциативными массивами или хеш-таблицами [7]. В основе реализации таких структур применяются методы хеширования, которые используются для более быстрого поиска элементов в массиве.

Графы могут быть очень большого размера, с миллиардами вершин. Процесс поиска каких-либо вершин в графах с подобной размерностью может стать очень медленным. В то же время при использовании хеширования, если сохранить информацию по уже известному вычисленному хеш-ключу (индексу), то в будущем можно будет всегда находить её за наименьшее (константное) время. Данные хранятся с использованием пар: «ключ ↦ значение». Каждому значению присваивается уникальный хеш-ключ (индекс), который вычисляется с помощью некоторой хеш-функции (f). Имя ключа (k) преобразуется в целое число (h) и используется в будущем для доступа к связанному с ним значению. Например: $h = f(k)$, где k – имя ключа, h – целое число (хеш-ключ), а f – некоторая хеш-функция.

Хеш-функция осуществляет отображение $k \mapsto h$, по некоторому закону. Например, ключ строки с названием «*FrFric*» может быть преобразован к её численному значению путем суммирования числовых кодов каждого символа и деления по модулю на какое-либо число:

$$h = (971 + 982 + 993 + 1004 + 1015 + 1026) \% 2069$$

В различных языках программирования алгоритмы хеширования реализованы по-разному для большей эффективности и избежания коллизий данных. В языке Python хеширование реализовано с использованием связных и двусвязных списков для дополнительного хранения порядка вставленных элементов.

Ниже приведен фрагмент кода, иллюстрирующий пример работы со структурой «OrderedDict» (листинг 1).

Листинг 1. Пример кода, использующего структуру «OrderedDict»

```
##
  G = OrderedDict()
  G['<FrFric>'] = [['f1','<FrPull>','<Mass>','<Accl>'],
  ['f12','<OpFric>', '<Dist>'],
  ['f3','<Cfg>','<CfFric>','<Mass>']]
  …
##
```

Приведенный пример показывает, что с вершиной «*FrFric*» связаны вершины «*FrPull*», «*Mass*», «*Accl*», «*Cfg*», «*CfFric*» и «*Mass*». Для удобства обработки информации, в дополнение к множествам, содержащим вершины графа, добавлены названия функций, принимающих данные элементы множеств, как аргументы для осуществления в последующем различных вычислений. Заданные функции, например, *f12*, также хранятся в базе знаний и описываются в виде лямбда-функций: *f12 = lambda x,y: (x / y).* Где x, y – аргументы лямбда-функции.

После заполнения базы знаний необходимой информацией и добавления всех необходимых функций для вычислений можно начинать работу с системой. Сам алгоритм поиска подграфа реализован в виде одной основной функции с именем «dfa», а также двух вспомогательных функций: «inputs» – для ввода информации и «buid» – для синтеза математических выражений и выполнения дальнейших вычислений. Ниже приведен фрагмент кода, реализующий ввод данных (листинг 2).

Листинг 2. Пример кода функции, реализующей ввод данных

```
##
  def inputs():
  S = input("Введите аксиому::")
  print("\r\n Введите данные в формате::
  {'<Accl>':0.3,'<Time>':20,'<Spd1>':4}")
  D = eval(input())
  VT = list(D.keys())
##
```

Для ввода информации необходимо использовать определенный формат:

```
{'<Accl>':0.3,'<Time>':20,'<Spd1>':4}
```

Он состоит из формальных названий терминальных вершин и принадлежащих им фактических числовых значений, взятых из постановки исходной задачи. Такой формат ввода был выбран не случайно, так как он чаще всего используется при передаче информации в сети Интернет между различными веб-сервисами.

Функция синтеза математических моделей работает в два этапа. В начале строится символьное представление модели, которое составляется из подмножеств вершин графа, заключаемых в скобки, в соответствии с определенным порядком. Затем вместо символьных обозначений вершин подставляются введенные фактические (численные) значения. Порядок подвыражений определяется в процессе поиска подграфа на этапе обхода вершин графа G. Для подробного ознакомления блок-схема работы основного алгоритма поиска подграфа доступна в виде электронного ресурса в сети Интернет [12].

6. Вычислительный эксперимент и результаты исследования. Проведем вычислительный эксперимент, подтверждающий работоспособность программной реализации описанного выше метода. Для этого решается задача из пособия под авторством Голубевой О.В. – «… За какое время автомобиль, двигаясь с ускорением 0,4 м/с^2, увеличит свою скорость с 12 до 20 м/с ? …» [13].

Для запуска программы на языке Python использован стандартный, официальный интерпретатор – CPython. Это бесплатный программный инструмент, исходный код которого доступен для изучения и загрузки на сайте GitHub [14]. В качестве операционной системы используется Windows 10.

Запуск программы осуществляется путем вызова интерпретатора и передачи ему файла с кодом программы через утилиту командной строки (cmd.exe). Далее запущенная программа просит ввести исходные данные по решаемой задаче: аксиому и известные величины в необходимом формате. Листинг 3 отображает процесс запуска и ввода данных в программу.

Листинг 3. Процесс запуска и ввода данных в программу

```
>
>python spF.py
>
Введите аксиому::
<Time>
Введите данные в формате:: {'<Accl>':0.3,'<Time>':20,'<Spd1>':4}
{'<Accl>':0.4,'<Spd1>':12,'<Spd2>':20}
>
```

После ввода необходимых обозначений величин (терминальных вершин) и соответствующих им числовых значений начинается процесс поиска слабо-связного подграфа. Если будет найден подграф, включающий в себя подмножество всех необходимых терминальных вершин, то будет выведено соответствующее сообщение. Далее все вершины подграфа выводятся на экран в виде линейной суперпозиции функций, в которой нетерминальные вершины заменяются именами ассоциативно связных с ними функций, а связанные с ними терминальные вершины обрамляются круглыми скобками. Таким образом, терминальные вершины становятся аргументами функций и при подстановке фактических числовых значений происходит вычисление результата. Вывод результатов решения задачи отображается ниже (листинг 4).

Листинг 4. Вывод результатов решения из программы

```
>
Подграф найден!
f8(<Spd2>,<Spd1>,<Accl>)
Ответ = 20.0
>
```

Если будут введены недопустимые данные или по введенным данным не удастся осуществить поиск подграфа ввиду неполноты базы знаний, то будет выведено сообщение об отсутствии решения.

Как можно видеть, результаты решения задач и ответы действительно соответствуют таковым. Работоспособность описанного выше метода и алгоритмов была проверена при решении десятков различных задач из области механики. Кроме того, был проведен сравнительный анализ эффективности алгоритмов реализации данного метода на разных языках программирования, который показал, что текущая реализация с использованием хеширования на языке Python более эффективна [15].

Заключение. В результате проделанной работы было разработано специальное математическое и алгоритмическое обеспечение, в виде программного модуля для систем анализа и обработки экспертной информации с целью автоматизированного поиска и синтеза решений задач моделирования и идентификации динамических систем.

Решена проблема поиска и логического вывода синтезированных решений задач на графе знаний. Представлена модель базы знаний выбранной предметной области в виде мультиграфа знаний. На основе теоретико-множественного анализа и элементов теории графов предложен новый модифицированный метод поиска и логического вывода синтезированных решений задач. Для демонстрации работоспособности предложенный метод реализуется программным алгоритмом, а алгоритм реализуется кодом на языке программирования Python.

Предложенные метод и алгоритмы могут быть использованы при практической реализации баз знаний, механизмов обработки и логического вывода экспертной информации в различного рода экспертных, расчетно-логических и гибридных интеллектуальных системах, замещающих собой зарубежные аналоги. Кроме того, с привлечением экспертов из разных предметных областей появится возможность составить новые базы знаний для решения большого круга других разнообразных задач. В подобных программных системах можно будет реализовать более удобный графический интерфейс для пополнения баз знаний, а также возможность их функционирования в сети Интернет.

Список источников

1. Кравченко В.А. Моделирование поиска решения с помощью функциональных грамматик / В.А. Кравченко // Вестник Бурятского государственного университета. Математика, информатика, 2012. – № 9. – С. 33-41.
2. Кравченко В.А. Формальное описание функционального логико-математического моделирования динамических систем / В.А. Кравченко, Д.Ш. Ширапов, Д.Н. Чимитов // Вестник Бурятского государственного университета. Математика, информатика, 2017. – № 3. – С. 32-39. – DOI:10.18101/2304-5728-2017-3-32-39.
3. Апанович З.В. Эволюция понятия и жизненного цикла графов знаний / З.В. Апанович // Системная информатика, 2020. – № 16. – С. 57-74. – DOI:10.31144/si.2307-6410

4. Бхатт Ш. Графы знаний как средство улучшения искусственного интеллекта / Ш. Бхатт, Ц. Чжао // Открытые системы, СУБД, 2020. – № 3. – С. 24-26.
5. Пилецкий И.И. Граф знаний и машинное обучение как базис методологии искусственного интеллекта в обучении / И.И. Пилецкий // Big Data and Advanced Analytics, 2021. – №7-1. – С. 198-210.
6. Лаврищева Е.М. Технология моделирования физических и математических задач предметных областей знаний / Е.М. Лаврищева // Евразийское Научное Объединение, 2021. – №1-1(71). – С. 35-43.
7. Кормен Т.Х. Алгоритмы: построение и анализ / Т.Х. Кормен, Ч.И. Лейзерсон, Р.Л. Ривест, К. Штайн // М.: «Вильямс», 2019. – 1328 с. – ISBN 978-5-9071-1411-1.
8. Седжвик Р. Алгоритмы на С++. Анализ структуры данных. Сортировка. Поиск. Алгоритмы на графах / Р.Седжвик // М.: «Вильямс», 2019. – 1056 с. – ISBN 978-5-8459-2070-6.
9. Promel H.J., Steger A. The Steiner tree problem: a tour through graphs, algorithms, and complexity. Springer Science & Business Media, 2012.
10. Pajor T., Uchoa E., Werneck R.F. A robust and scalable algorithm for the Steiner problem in graphs. Math. Program. Comput, 2017, DOI:10.1007/s12532-017-0123-4.
11. Van Rossum G. Python Tutorial: Release 3.6.4. Pub: 12th Media Services, 2018, 156 p., ISBN-13:978-1680921601.
12. Зайцев А.Ф. Flowchart of the main algorithm for finding a subgraph in a directed, unweighted, cyclic multigraph. – URL: https://github.com/Toljanchiman/lispVSpython/blob/master/img/PyFlowChart.png (дата обращения: 13.06.2022).
13. Голубева О.В. Механика: учебное пособие / О.В. Голубева, С.Г. Жигаленко, О.И. Гаммершмидт // Липецк: Липецкий ГПУ, 2018. – 99 с. – ISBN 978-5-88526-928-5
14. Van Rossum G. The Python programming language interpreter. Available at: https://github.com/python/ cpython (accessed: 13.06.2022).
15. Зайцев А.Ф. Анализ эффективности алгоритмов вычисления в информационной системе логико-математического моделирования / А.Ф. Зайцев, В.А. Кравченко, Д.Ш. Ширапов // Вестник Бурятского государственного университета: Математика, информатика, 2020. – № 2. – С. 3-14. – DOI:10.18101/2304-5728-2020-2-3-14.

Зайцев Анатолий Федорович. *Преподаватель кафедры электронно-вычислительных систем, ФГБОУ ВО Восточно-Сибирский государственный университет технологий и управления, AuthorID: 1037054, SPIN: 9938-9462, ORCID: 0000-0002-0960-9061, lordsadler2010@mail.ru, Россия, Улан-Удэ, ул. Смолина 26.*

UDC 004.82
DOI:10.38028/ESI.2022.28.4.017

Method of search and logical inference of expert information in a directed cyclic knowledge multigraph

Anatoly F. Zaytsev

East Siberian state university of technology and management,
Russia, Ulan-Ude, *lordsadler2010@mail.ru*

Abstract. Currently, many modern problems of applied mathematics and computer science are solved using graph theory. Various complex systems, such as neural networks or knowledge bases, can be represented and described as graphs. The most common problems using graph theory are: finding the shortest path, determining the maximum flow in the network, finding the minimum spanning trees and others. At the same time, there are quite a lot of unresolved problems. The relevance of the work is due to increased interest in the fields of artificial intelligence and knowledge engineering, the methods of which are the possibility of transforming the obtained subject models into logical and mathematical, in the form of programs for computers that carry out computer or simulation modeling of the systems under study. In the presented work the process of development of the special mathematical and algorithmic software for system of the analysis and processing of the expert information for the purpose of the automated search and modeling of decisions of problems for identification of dynamic systems is described. The problem of search and logical inference of synthesized solutions of problems on knowledge graphs is formulated. The model of knowledge base of the selected subject area in the form of knowledge multigraph is presented, as well as the method of search and logical inference of solutions of problems, with their software implementation in the programming language Python. The presented method of search of weakly connected subgraphs and synthesis of problem solutions is implemented by using theoretical-multiple analysis, as well as elements of graph theory. To demonstrate the performance of the method, its implementation in the form of an algorithm in the Python programming language and the results of computational experiments are given. The novelty and practical significance of the work lies in the fact that the proposed method and algorithms can be used in the practical implementation of knowledge bases, logical inference mechanisms and processing of expert information in a variety of expert, calculation-logical and hybrid intelligent systems, replacing their foreign analogues.

Keywords: system analysis, logical inference, graph search, multigraph, expert system, knowledge base, knowledge graph, Python

References

1. Kravchenko V.A. Modelirovanie poiska reshenija s pomoshh'ju funkcional'nyh grammatik [Modeling the search for solutions using functional grammars]. Vestnik Buryatskogo gosudarstvennogo universiteta. Matematika, informatika [Bulletin of the Buryat State University. Mathematics Informatics], 2012, no. 9, pp. 33-41.
2. Kravchenko V.A., Shirapov D. Sh., Chimitov D.N. Formal'noe opisanie funkcional'nogo logiko-matematicheskogo modelirovanija dinamicheskih sistem [Formal description of functional logical and mathematical modeling of dynamic systems]. Vestnik Buryatskogo gosudarstvennogo universiteta. Matematika, informatika [Bulletin of the Buryat State University. Mathematics Informatics], 2017, no. 3, pp. 32-39, DOI:10.18101/2304-5728-2017-3-32-39.
3. Apanovich Z.V. Jevoljucija ponjatija i zhiznennogo cikla grafov znanij [Evolution of the concept and life cycle of knowledge graphs]. Sistemnaya informatika [System Informatics], 2020, no. 16, pp. 57-74, DOI:10.31144/si.2307-6410.
4. Bkhatt SH., CHzhao TS. Grafy znanij kak sredstvo uluchshenija iskusstvennogo intellekta [Knowledge graphs as a means to improve artificial intelligence]. Otkrytyye sistemy, SUBD [Open Systems DBMS], 2020, no.3, pp. 24-26.
5. Piletskij I.I. Graf znanij i mashinnoe obuchenie kak bazis metodologii iskusstvennogo intellekta v obuchenii [The Knowledge Graph and Machine Learning as the Basis for Artificial Intelligence Methodology in Learning]. Big Data and Advanced Analytics, 2021, no. 7-1, pp. 198-210.
6. Lavrishheva E.M. Tehnologija modelirovanija fizicheskih i matematicheskih zadach predmetnyh oblastej znanij [Technology of modeling physical and mathematical problems of subject areas of knowledge]. Yevraziyskoye Nauchnoye Ob"yedineniye [Eurasian Scientific Association], 2021, no. 1-1(71), pp. 35-43.
7. Kormen T.KH., Lejzerson CH.I., Rivest R.L., SHtajn K. Algoritmy: postroenie i analiz [Algorithms: construction and analysis. 3rd ed]. M.: Williams, 2019, 1328 p., ISBN 978-5-9071-1411-1.
8. Sedzhvik R. Analiz struktury dannyh. Sortirovka. Poisk. Algoritmy na grafah [Algorithms in C++. Data structure analysis. Sorting. Search. Algorithms on graphs]. Moscow: Williams, 2019, 1056 p., ISBN 978-5-8459-2070-6.
9. Promel H.J., Steger A. The Steiner tree problem: a tour through graphs, algorithms, and complexity. Springer Science & Business Media, 2012.
10. Pajor T., Uchoa E., Werneck R.F. A robust and scalable algorithm for the Steiner problem in graphs. Math. Program. Comput, 2017, DOI:10.1007/s12532-017-0123-4
11. Van Rossum G. Python Tutorial: Release 3.6.4. Pub: 12th Media Services, 2018, 156 p., ISBN-13:978-1680921601.
12. Zaytsev A.F. Flowchart of the main algorithm for finding a subgraph in a directed, unweighted, cyclic multigraph [Electronic resource]. Available at: https://github.com/Toljanchiman/lispVSpython/blob/master/ img/ Py-Flow- Chart.png (accessed: 13.06.2022).
13. Golubeva O.V. Mekhanika: uchebnoe posobie [Mechanics: tutorial]. Lipetsk: Lipetskij GPU, 2018, 99 p., ISBN 978-5-88526-928-5
14. Van Rossum G. The Python programming language interpreter. Available at: https://github.com/python/cpython (accessed: 13.06.2022).
15. Zaytsev A.F., Kravchenko V.A., Shirapov D.Sh. Analiz jeffektivnosti algoritmov vychislenija v informacionnoj sisteme logiko-matematicheskogo modelirovanija [Analysis of the efficiency of calculation algorithms in the information system of logical and mathematical modeling]. Vestnik Buryatskogo gosudarstvennogo universiteta. Matematika, informatika [Bulletin of the Buryat State University. Mathematics Informatics], 2020, no. 2, pp. 3-14, DOI:10.18101/2304-5728-2020-2-3-14.

Anatoly Fedorovich Zaytsev. *Lecturer, department of electronic-computing systems, East Siberian State University of Technology and Management, AuthorID: 1037054, SPIN: 9938-9462, ORCID: 0000-0002-0960-9061, lordsadler2010@mail.ru, Russia, Ulan-Ude, 26 Smolina St.*

Статья поступила в редакцию 15.06.2022; одобрена после рецензирования 06.10.2022; принята к публикации 01.11.2022.
The article was submitted 06/15/2022; approved after reviewing 10/06/2022; accepted for publication 11/01/2022.