2. Забара С. С., Изварин И. В. Семантические базы данных документов с различным реквизитным содержанием // Комп'ютерно-інтегровані технології: освіта, наука, виробництво: Науковий журнал / Під редакцією доктора технічних наук В. Д. Рудь. – Луцьк: Видавництво Луцького національного технічного університету, 2011, № 5. — С. 87-92.
3. Изварин И.В. Жизненный цикл информационного потока // Интеллектуальные сисьемы принятия решений и проблемы вычислительного интеллекта: Материалы международной научной конференции. Том 1. – Херсон: ЧНТУб 2011ю – с. 76-78.
4. Gerald J. Kowalski, Mark T. Maybury. Information Storage and Retrieval Systems. Theory and Implementation. Second Edition. Kluwer Academic Publishers 2002.
5. Greengrass Ed. Information Retrieval: A Survey, 2000.
6. Stefano Cen, Marco Brambilla. Search Computing. Trends and Developments. Springer-Verlag Berlin Heidelberg 2011.

На основі фреймової моделі подання знань запропоновано принципи розробки баз знань та експертних систем універсальною мовою програмування Python. Проаналізовано об'єктно-орієнтовані можливості та засоби інтроспекції Python. Розроблено демонстраційну базу знань та подано приклади запитів до неї

Ключові слова: база знань, онтологія, подання знань, експертна система

На основе фреймовой модели представления знаний предложены принципы разработки баз знаний и экспертных систем на универсальном языке программирования Python. Проанализированы объектно-ориентированные возможности и средства интроспекции Python. Разработана демонстрационная база знаний и даны примеры запросов к ней

Ключевые слова: база знаний, онтология, представление знаний, экспертная система

On the basis of frames for knowledge representation have been proposed principles for the development of knowledge bases and expert systems on general-purpose programming language Python. Object-oriented and introspection capabilities of Python have been analyzed. The demo of knowledge base and the examples of querying to it have been developed

Keywords: knowledge base, ontology, knowledge representation, expert system

УДК 004.822

ЗАСТОСУВАННЯ МОВИ ПРОГРАМУВАННЯ PYTHON ДЛЯ ПОБУДОВИ БАЗ ЗНАНЬ ТА ЕКСПЕРТНИХ СИСТЕМ

В.Б. Копей
Кандидат технічних наук, доцент
Кафедра технології нафтогазового машинобудування
Івано-Франківський національний технічний університет нафти і газу
вул. Карпатська, 15, м. Івано-Франківськ, 76019
Контактний тел.: (03422) 4-30-24
E-mail: vkopey@gmail.com

Л.М. Семанишин
Аспірант
Івано-Франківська філія Відкритого міжнародного університету розвитку людини «Україна»
вул. Набережна, 42а, м. Івано-Франківськ, 76010
Контактний тел: 096-990-89-38
E-mail: symanyshyn_lesja@mail.ru

1. Вступ

Відомо, що розвиток науки основується на отриманні об'єктивних і системно-організованих **знань**, які допомагають людям раціонально організувати свою діяльність і вирішувати різні проблеми, котрі виникають в її процесі. Сучасні комп'ютерні технології зумовили розвиток **інженерії знань** - області науки про **штучний інтелект**, яка вивчає методи і засоби отримання, представлення, структурування і використання знань. Інженерія знань пов'язана з розробкою баз знань і експертних систем [1,2].

База знань - це сукупність **фактів** і **правил логічного висновку (виведення)** в обраній предметній області. **Правила логічного висновку** - це правила перетворення вихідної системи фактів (суджень) в нову систему фактів (висновків). Наприклад, за такими правилами з вихідних фактів "всі люди смертні" і "Сократ - людина" можна зробити висновок "Сократ смертний". В даному випадку правилом висновку є

параметризоване твердження “якщо всі люди смертні і X - людина, то X смертний”. Власне наявність правил висновку є основною відмінністю баз знань від баз даних.

Програма, яка виконує логічний висновок з попередньо побудованої бази фактів і правил у відповідності з законами формальної логіки, називається **машиною виведення**. База знань і машина виведення є основними компонентами **експертної системи** - програми, яка призначена для пошуку способів вирішення проблем з певної предметної області. Для такого пошуку користувач подає відповідні **запити до бази знань**.

Для збереження знань в комп’ютерній системі з можливістю їх подальшої автоматизованої обробки використовують різноманітні методи **подання (представлення) знань**. Прикладами методів подання чітких знань є: логіка першого порядку, мова програмування Пролог, реляційні системи, продукційна модель, фреймова модель, семантичні мережі [1,2]. Ці методи можна реалізувати різними формальними мовами подання знань, наприклад класу онтологій. Під **онтологією** в інформатиці розуміють детальну формалізацію деякої області знань за допомогою концептуальної схеми. Онтологія може використовуватись для представлення в базі знань ієрархії понять і їх відносин. Наприклад, мова Веб-Онтологій OWL (ontology web language) [3] була розроблена для реалізації концепції семантичної павутини в межах всесвітньої павутини (WWW) і є розширенням мови RDF (Resource Description Framework) - мови розмітки на основі XML, яка використовується для подання тверджень про ресурси в вигляді придатному для машинної обробки. Діалекти OWL Lite і OWL DL основані на **дескріпційних логіках** - мовах подання знань, що дозволяють описувати поняття предметної області в недвозначному, формалізованому вигляді. Для створення запитів до даних, поданих за моделлю RDF, використовується мова запитів SPARQL. Не зважаючи на усі переваги OWL і SPARQL є спеціальними мовами, що певною мірою обмежує їх можливості.

2. Постановка задачі

Основною ідеєю авторів є застосування в якості мови **подання знань** та **створення запитів** до бази знань мови програмування Python [4,5], яка на відміну від спеціальних мов є мовою загального призначення, і тому дає розробнику експертних систем набагато більше можливостей. Для цього можна використати **об’єктно-орієнтовані можливості** Python та її **засоби інтроспекції**, адже відомо, що такі методи подання чітких знань як фрейми і семантичні мережі можна просто реалізувати засобами **об’єктно-орієнтованого програмування** (ООП).

3. Огляд можливостей Python для подання знань

Чому авторами була вибрана саме Python? Python - розповсюджена високорівнева мова загального призначення з акцентом на продуктивність розробника. Основні її особливості: працює майже на усіх відомих платформах, є відкритим і вільним програмним забезпеченням, виконується шляхом інтерпретації байт-коду, підтримує кілька парадигм програмування (в тому числі об’єктно-орієнтоване), код програм компактний і легко читається, мові характерні динамічна типізація, повна інтроспекція, зручні структури даних (кортежі, списки, словники, множини), велика стандартна бібліотека та велика кількість сторонніх бібліотек різноманітного призначення.

Python володіє потужними об’єктно-орієнтованими можливостями, наприклад, усі значення в програмі є об’єктами. ООП основане на використанні **об’єктів**, які є абстрактними моделями реальних об’єктів. Об’єкти створюються за допомогою спеціальних типів даних - класів. Кожен **клас** описує множину об’єктів певного типу. Приклад ООП мовою Python:

```
# -*- coding: UTF-8 -*-
import inspect

# клас A успадкований від object
class A(object):
    '''Клас A''' # рядок документації
    def __init__(self,a): # конструктор
        self.a=a # атрибут a
    def f(self): # метод f
        '''Повертає self.a''' # рядок документації
        return self.a
class B(A): # клас B успадкований від A
    '''Клас B''' # рядок документації
    def __init__(self,a,b): # конструктор
        # виклик конструктора базового класу
        super(B, self).__init__(a)
        self.b=b # атрибут b
    def f(self): # метод f
        '''Повертає суму self.a+self.b''' # рядок документації
        return self.a+self.b

obj=B(0,2) # створення об'єкта класу B, виклик конструктора
obj.a=1 # зміна значення атрибута a
print obj.f() # виклик методу f
```

Основними принципами ООП є інкапсуляція, успадкування і поліморфізм. **Інкапсуляція** - об’єднання даних (атрибутів) і функцій їх опрацювання (методів) в класі.

Наприклад, в класі B об’єднано атрибути a і b та методи __init__() і f(). **Успадкування** - можливість створення похідних класів шляхом успадкування ними членів базового класу. Наприклад, атрибут a класу B успадкований від класу A. **Поліморфізм** - властивість об’єктів з однаковою специфікацією мати різну реалізацію. Наприклад, функції f() класів A і B мають одну назву, але виконується по-різному.

Розглянемо **засоби інтроспекціїї** Python. Python підтримує повну інтроспекцію часу виконання. Тобто для будь-якого об’єкта під час виконання можна отримати всю інформацію про його структуру. Приклади:

```
print dir(B) # список імен атрибутів класу B
print dir(obj) # список імен атрибутів об'єкта obj

print id(obj) # унікальний ідентифікатор об'єкта
print obj.__sizeof__() # розмір об'єкта в пам'яті в байтах

print B.__doc__ # рядок документації класу B
print obj.f.__doc__ # рядок документації методу f

print B.__name__ # ім'я класу B
print __name__ # ім'я модуля

print type(obj) # тип (клас) об'єкта obj
# або
print obj.__class__
print obj.__class__.__name__ # ім'я типу об'єкта obj

print vars(obj) # словник з парами атрибут:значення
# або
print obj.__dict__
print hasattr(obj, 'a') # чи є атрибут 'a' у об'єкта obj?
setattr(obj,'a',3) # зміна значення (3) атрибута 'a' об'єкта obj
# або
obj.__setattr__('a',3)
print getattr(obj, 'a') # значення атрибута 'a' об'єкта obj
# або
print obj.__getattribute__('a')
# або
print obj.__dict__['a']

print callable(obj.f) # чи атрибут f є методом?

print isinstance(obj, B) # чи obj є екземпляром B?

print issubclass(A, object) # чи A є підкласом object?
print A.__subclasses__() # підкласи A
print B.__bases__ # кортеж базових класів
print B.__mro__ # кортеж з ієрархією базових класів
# або
import inspect
print inspect.getmro(B)

# повертає список пар (ім'я, значення) членів об'єкта
print inspect.getmembers(obj)
print inspect.getcomments(A) # коментар перед класом A
print inspect.getsource(A) # текст вихідного коду класу A
print inspect.isclass(A) # чи A є класом?
print inspect.ismethod(obj.f) # чи obj.f є методом?
```

Найбільш відомим інструментом для інтроспекції в Python є функція dir(), яка повертає список імен атрибутів переданого їй об'єкта. Ця функція працює з усіма типами об'єктів. Функція id() повертає унікальний ідентифікатор об'єкта. Рядок документації _doc_ - це атрибут, який містить рядок-коментар, який описує об'єкт. Рядки документації дозволяють зберігати документацію на класи і функції прямо в коді програми і звертатись до неї під час виконання програми. Атрибут _name_ містить ім'я об'єкта. Функція type() або атрибут _class_ дозволяють отримати тип об'єкта. Функція vars() або атрибут _dict_ дозволяють отримати словник з парами атрибут: значення об'єкта. Функції hasattr(), getattr() і setattr() дозволяють відповідно перевірити наявність у об'єкта заданого атрибута, повернути його і змінити значення. Функція callable() дозволяє визначити чи об'єкт є функцією або методом. Функція isinstance() дозволяє визначити чи об'єкт є екземпляром заданого типу. Функція issubclass() дозволяє визначити чи успадковується один клас від іншого, а метод _subclasses_() повертає список підкласів. Кортеж базових класів та їх ієрархію можна отримати за допомогою атрибутів _bases_ і _mro_. Модуль inspect містить додаткові функції, які допомагають отримати інформацію про об'єкти під час виконання.

Застосування інтроспекції дозволяє ефективно програмувати механізми виведення та запити до бази знань.

4. Приклад бази знань і запитів до неї мовою Python

Подамо приклад створення мовою Python демонстраційної бази знань, яка містить поняття (класи) персони і сім'ї, відповідні індивіди (об'єкти) та властивості (атрибути), які описують родинні зв'язки між індивідами. Для цього застосуємо фреймову модель подання знань.

Елементи розробленої авторами онтології частково відповідають основним елементам OWL Lite [3]: класи (з можливістю створення підкласів), властивості (які можуть бути функціональними, інверсними, симетричними і транзитивними) і індивіди. Тому можливим є імпорт і експорт таких онтологій в OWL. Модуль Python, який містить базу знань і запити до неї повинен мати блочну структуру - спочатку створюються класи, потім індивіди, потім задаються властивості індивідів, а в кінці виконуються запити.

Класи онтології відповідають класам мови Python, а підкласи можна створювати за допомогою механізму успадкування Python. Так розроблено базовий клас Base (описує усі об'єкти, які мають ім'я), класи Persona (описує персону), PersonaMarried (описує одружену персону, успадковується від Persona) і Family (описує сім'ю).

```
# -*- coding: UTF-8 -*-
class Base(object):
  '''Базовий клас онтології'''
  def __init__(self,name):
    self.name=name # назва об'єкта
    KB[self.name]=self # добавити себе в базу знань
##################################################################
class Persona(Base): # успадковує клас Base
  '''Клас, який описує персону'''
  def __init__(self,name):
    Base.__init__(self,name) # виклик конструктора базового класу
    # симетрична і транзитивна властивість 'є братом/сестрою'
    Property(subj=self, name='isSibling',
        symmetric=True, transitive=True)
    # транзитивна властивість 'є нащадком' (інверсна - 'є батьком')
    Property(subj=self, name='isChild',
        inverseName='isParent', transitive=True)
    # властивість 'є членом сім'ї' (інверсна - 'має членів')
    Property(subj=self, name='isMember',
        inverseName='hasMembers',transitive=True)
##################################################################
class PersonaMarried(Persona):
```

```
    '''Клас, який описує одружену персону'''
    def __init__(self,name):
        Persona.__init__(self, name)
        # симетрична і функціональна властивість 'є одружений'
        Property(subj=self, name='isMarried',
                 symmetric=True, functional=True)
        # транзитивна властивість 'є батьком' (інверсна - 'є нащадком')
        Property(subj=self, name='isParent',
                 inverseName='isChild', transitive=True)
##########################################################
class Family(Base):
    '''Клас, який описує сім'ю'''
    def __init__(self,name):
        Base.__init__(self,name)
        # властивість 'має членів' (інверсна - 'є членом сім'ї')
        Property(subj=self, name='hasMembers',
                 inverseName='isMember', transitive=True)
    def number(self):
        '''Повертає кількість членів сім'ї'''
        return self.hasMembers().__len__()
```

Індивіди онтології відповідають об’єктам Python - значенням словника КВ. Такий спосіб дозволяє звертатись до індивідів за їх іменем у вигляді Юнікод-рядка. Індивіди можуть мати властивості у вигляді атрибутів Python, які дозволяють описувати відношення між індивідами. Властивості є об’єктами класу Property, мають атрибути subj (суб’єкт властивості - це індивід, який має дану властивість), name (назва властивості), inverseName (назва інверсної властивості), functional (визначає чи властивість функціональна), symmetric (визначає чи властивість симетрична), transitive (визначає чи властивість транзитивна), set (множина значень властивості) та методи _init_() (конструктор, створює властивість), add() (добавляє об’єкт або кортеж об’єктів в множину) і _call_() (повертає множину значень властивості).

Останні два методи реалізують елементи машини виведення. Такі елементи можуть бути присутні також в запитах до бази знань.

```
class Property(object):
    '''Клас, який описує властивість об'єкта'''
    def __init__(self,subj,name,inverseName='',functional=False,
                 symmetric=False, transitive=False):
        '''Конструктор'''
        self.subj=subj # суб'єкт властивості
        self.name=name # назва властивості
        self.inverseName=inverseName # назва інверсної властивості
        self.functional=functional # властивість функціональна
        self.symmetric=symmetric # властивість симетрична
        self.transitive=transitive # властивість транзитивна
        self.set=set() # множина значень властивості
        # установити атрибут властивості для суб'єкта
        self.subj.__setattr__(self.name,self)
    def add(self,*args):
        '''Добавляє об'єкт або кортеж об'єктів в множину'''
        for obj in args: # для всіх об'єктів в args
            if self.functional: # якщо властивість функціональна
                self.set.clear() # очистити множину

            if self.inverseName!='': # якщо є інверсна властивість
                # якщо суб'єкта немає в інверсній властивості об'єкта
                if self.subj not in obj.__dict__[self.inverseName].set:
                    # добавити його в інверсну властивість об'єкта
                    obj.__dict__[self.inverseName].set.add(self.subj)

            if self.symmetric: # якщо властивість симетрична
                #якщо суб'єкта немає в аналогічній властивості об'є
                if self.subj not in obj.__dict__[self.name].set:
                    # добавити його в аналогічну властивість об'єкта
                    obj.__dict__[self.name].set.add(self.subj)

            self.set.add(obj) # добавити об'єкт в множину
    def __call__(self, showTransitive=False):
        '''Повертає множину значень властивості
        (showTransitive=True - транзитивної властивості)
        Дозволяє викликати об'єкт як функцію'''
        def getTransitive(subj,s=set()):
            '''Повертає множину значень транзитивної
            властивості. Рекурсивна'''
            # якщо subj має атрибут self.name
            if hasattr(subj, self.name):
                #для усіх об'єктів в властивості з назвою self.nam
                for obj in subj.__dict__[self.name].set:
                    if obj not in s: # якщо obj немає в множині s
                        s.add(obj) # добавити об'єкт в множину
                        s=getTransitive(obj,s) # рекурсія
            return s # повертає множину
        # якщо властивість транзитивна і показувати транзи
        if self.transitive and showTransitive:
            # множина значень транзитивної властивості
            s=getTransitive(self.subj)
            # вилучити self.subj з множини, якщо є
            s.discard(self.subj)
            return s
        # інакше повернути множину значень властивості
        else: return self.set
```

База знань містить індивіди ‘я’, ‘дружина’, ‘мати’, ‘батько’, ‘донька’ і т.д. - об’єкти відповідних класів. Створений об’єкт індивіда автоматично додається в словник КВ. Блок створення об’єктів індивідів виглядає так:

```
KB={} # словник бази знань
Persona('онук')
Persona('племінниця')
Persona('син')
for x in ('я','дружина','мати','батько','донька','зять',
          'теща','тесть','дід','баба','стрийко','прадід',
          'брат','сестра','вуйко','тітка'):
    PersonaMarried(x)
```

Клас Persona містить властивості isSibling (є братом/сестрою), isChild (є нащадком), isMember (є членом сім’ї); клас PersonaMarried - властивості isMarried (є одруженим з), isParent (є батьком); клас Family - властивість hasMembers (має членів). Властивість створюється під час створення індивіда шляхом виклику її конструктора _init_() з параметрами subj, name, inverseName, functional, symmetric, transitive, які описують її характеристики. Так властивість isMarried є **функціональною**, тобто має унікальне значення (адже неможливо бути одруженим більше ніж з одним індивідом). Вона також є **симетричною**, що дозволятиме, наприклад, з твердження “я одружений з дружиною” робити логічний висновок “дружина одружена зі мною”. Властивості isChild і isParent є взаємно **інверсними**. Це дозволяє, наприклад, з твердження “я є нащадком діда” робити логічний висновок “дід є моїм предком”. Вони також є **транзитивними**. Це дозволяє, наприклад, з тверджень “я є нащадком батька”

і “батько є нащадком діда” робити логічний висновок “я є нащадком діда”.

Значення в властивість добавляються за допомогою методу add(). Блок добавлення значень в властивості індивідів виглядає так:

```
# суб'єкт 'я' 'є одружений' з об'єктом 'дружина'
KB['я'].isMarried.add(KB['дружина'])
KB['я'].isChild.add(KB['мати'],KB['батько'])
KB['я'].isParent.add(KB['син'],KB['донька'])
KB['я'].isSibling.add(KB['брат'],KB['сестра'])
KB['мати'].isChild.add(KB['баба'],KB['дід'])
KB['зять'].isParent.add(KB['онук'])
KB['зять'].isChild.add(KB['теща'],KB['тесть'])
KB["моя сім'я"].hasMembers.add(KB['я'],KB['дружина'],
                    KB['син'],KB['донька'])
...
```

Запити до бази знань створюються шляхом пошуку об’єктів в множинах. Для цього можна використовувати оператори Python for та if. Для доступу до множин використовують об’єкт KB, метод класу Property _call_() та стандартні математичні операції над множинами. Метод _call_()з параметром showTransitive=True повертає множину усіх значень транзитивної властивості. Наприклад, запит, який виводить усіх предків онука:

```
for x in KB['онук'].isChild(showTransitive=True):
   print x.name,
```

На відміну від OWL в класах такої онтології можуть бути властивості, які не зберігають конкретного значення, а дозволяють його обчислити. Наприклад, метод класу Family number() обчислює і повертає кількість членів сім’ї. Наступний запит виводить сім’ї, в яких більше двох членів:

```
for x in KB.itervalues():
   if x.__class__.__name__=='Family':
      if x.number()>2:
         print x.name,
```

Серед вихідних фактів відсутні факти про батьків брата і сестри, але є факти про моїх батьків, брата і сестру. Тому наступний запит дозволяє вивести батьків брата і сестри:

```
for x in KB['брат'].isSibling(True)|KB['сестра'].isSibling(True):
   for y in x.isChild():
      print y.name,
```

Перед блоком запитів можна створити блок з правилами виведення. Для прикладу, створимо правило виведення: “якщо x є братом (сестрою) y і x є дитиною z, то y є дитиною z “.

```
p=[] # обмежимо множину пошуку класом PersonaMarried
for x in KB.itervalues():
   if x.__class__.__name__=='PersonaMarried':
      p.append(x)
for x in p:
   for y in p:
      for z in p:
         if x in y.isSibling() and x in z.isParent():
            # добавити нові факти в базу знань
            KB[y.name].isChild.add(KB[z.name])
            print y.name+'-'+z.name,
```

Наявність великої кількості таких правил може суттєво уповільнити виконання програми, тому бажано виконувати цей код тільки один раз і зберігати об’єкти бази знань в постійній пам’яті за допомогою способів серіалізації даних Python (модулів pickle або shelve).

5. Висновки

Запропоновані авторами методи побудови баз знань і експертних систем мовою Python мають переваги у порівнянні з існуючими. Основною перевагою є те, що розробнику доступні усі широкі можливості мови Python.

Під час розширення функціональності системи немає потреби винаходити ще одну спеціальну мову опису знань і мову запитів до них. Завдяки універсальності Python база знань є гнучкою до змін, існує можливість легкого удосконалення класів, атрибутів, правил виведення і запитів. Ці принципи можна використати для розробки повноцінних експертних систем в науці і техніці. Наприклад, автор планує їх застосування для побудови експертної системи з проблем надійності і довговічності різьбових з’єднань [6].

В такому підході до побудови баз знань є і певні недоліки. Зокрема, користувач повинен добре володіти мовою Python. Недоліком є також складність програмування машини виведення. Проте, якщо онтологія може бути експортована в OWL, то до програми можна підключати відомі машини введення FaCT++, HermiT, Cwm (реалізована на Python), Pellet, які мають ефективні алгоритми і доступні у вихідному коді. У порівнянні з існуючими редакторами онтологій тут можуть бути певні незручності створення великої бази знань.

Тому для ефективного використання програма потребує надбудови у вигляді графічного інтерфейсу користувача.

Література

1. Субботін С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень: навчальний посібник / С. О. Субботін. — Запоріжжя: ЗНТУ, 2008. — 341 с.
2. Рыбина Г.В. Основы построения интеллектуальных систем / Г. В. Рыбина. - М.: Финансы и статистика, ИНФРА-М, 2010. - 432 с.
3. OWL Web Ontology Language. Overview [Електронний ресурс]: W3C Recommendation 10 February 2004 / W3C. - Режим доступу: http://www.w3.org/TR/owl-features/.

4. Бизли Д. Python. Подробный справочник / Дэвид Бизли, 4-е издание. — Пер. с англ. — СПб.: Символ-Плюс, 2010. — 864 с
5. Лутц М. Изучаем Python / Марк Лутц, 4-е издание - Пер. с англ. - СПб.: Символ-Плюс, 2010. — 1280 с.
6. Копей В.Б. Принципи розробки бази знань з проблем надійності і довговічності різьбових з'єднань / В.Б. Копей, Ю.Д. Петрина // Науковий вісник Національного технічного університету нафти і газу. - № 4(26). - 2010. - С.66-69.

Викладена методика експериментального визначення розподілу ймовірностей значень параметрів транспортних потоків на вулично-дорожній мережі міста

Ключові слова: емпірико-стохастичний підхід, розподіл ймовірностей

Изложена методика экспериментального определения распределения вероятностей значений параметров транспортных потоков на улично-дорожной сети города

Ключевые слова: эмпирико-стохастический подход, распределение вероятностей

The method of experimental determination of the probability distribution of the parameters of traffic flows on the road network of the city are presented

Keywords: empirical-stochastic approach, probability distribution

УДК 656.13

МЕТОДИКА ЭКСПЕРИМЕНТАЛЬНОГО ОПРЕДЕЛЕНИЯ РАСПРЕДЕЛЕНИЯ ПАРАМЕТРОВ ТРАНСПОРТНЫХ ПОТОКОВ

Е.М. Гецович
Доктор технических наук, профессор*

Н.А. Семченко
Старший преподаватель*
*Кафедра организации и безопасности дорожного движения
Харьковский национальный автомобильно-дорожный университет
ул. Петровского, 25, г. Харьков, Украина, 61002
Контактный тел.: (057) 707-37-06

1. Вступление

Непрерывный рост уровня автомобилизации населения при существенном отставании развития улично-дорожной сети (УДС) городов в сочетании с высокой концентрацией центров тяготения в их центральных деловых частях (ЦДЧМ) обусловливают высокий спрос на въезд, проезд и парковку в ЦДЧГ. Одним из наиболее эффективных путей снижения напряженности в работе транспортной системы ЦДЧГ является создание автоматической системы управления дорожным движением (АСУ ДД) всей центральной части. В задачи такой системы входит непрерывный мониторинг транспортных потоков, прогнозирование развития ситуаций на всех участках УДС ЦДЧГ, выявление проблемных участков и перекоммутация транспортных потоков в обход проблемных участков с помощью различных технических средств регулирования дорожного движения (ДД).

Решение задачи прогнозирования развития ситуаций на всех участках УДС ЦДЧГ возможно только на основе моделирования движения транспортных потоков при заданной схеме организации ДД.

Авторам представляется возможным создание модели движения транспортных потоков и на ее базе – программного продукта на основе эмпирико-стохастического подхода, который позволяет в сотни раз уменьшить количество уравнений и условий и при этом сохранить точность описания движения и возможность рассмотрения дискретно-непрерывных транспортных потоков, свойственные микроскопическим моделям.

При эмпирико-стохастическом подходе транспортный поток представляется в виде движущихся «пакетов» автомобилей [1].

Характеристики потока в этом случае (число автомобилей в пакете, расстояние между пакетами, динамический габарит автомобиля в пакете, габарит пакета) носят случайный характер и при моделировании движения потока могут задаваться каким-либо распределением вероятности. Распределение вероятности по каждому параметру подлежит экспериментальному определению путем обследования реальных транспортных потоков на улично-дорожной сети (УДС) города. При этом следует учитывать, что закон распределения и расположение дисперсий по оси абсцисс перечисленных параметров зависят от интенсивности потока. Интенсивность потока подлежит определению при мониторинге транспортных потоков на УДС.