СОДЕРЖАНИЕ

ТЕХНИКА, ТЕХНОЛОГИЯ, УПРАВЛЕНИЕ, КВАНТОВЫЕ ТЕХНОЛОГИИ

По материалам XXI Международной научно-технической конференции «Новые информационные технологии и системы» (НИТиС-2024), посвященной 50-летнему юбилею кафедры «Математическое обеспечение и применение электронных вычислительных машин».

ТЕХНИКА, ТЕХНОЛОГИЯ, УПРАВЛЕНИЕ, КВАНТОВЫЕ ТЕХНОЛОГИИ

УДК 004.432.2

СРАВНЕНИЕ КОМПИЛЯТОРА И ИНТЕРПРЕТАТОРА НА ПРИМЕРЕ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ С И PYTHON

А. Р. Акчурин[1], К. С. Симакин[2], О. С. Дорофеева[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1]jkrez.zip@mail.ru
[2]simakinkonstantin@gmail.com
[3]dos@pnzgu.ru

Аннотация. Компилятор – это программа, которая преобразует исходный код в машинный, который может выполняться процессором компьютера. Он обрабатывает код последовательно через несколько стадий: анализирует, оптимизирует и создает исполняемый файл. Это позволяет программе, написанной на языке высокого уровня, работать на вычислительной машине. Интерпретатор – это программа, которая выполняет исходный код построчно, переводя его в машинный код в процессе исполнения программы, вместо того чтобы заранее компилировать код в исполняемый файл. Выполнен сравнительный анализ работы компилятора и интерпретатора на примере языков программирования С и Python.

Ключевые слова: компилятор, интерпретатор, байт-код, С, Python, препроцессинг, ассемблирование, линковка

Для цитирования: Акчурин А. Р., Симакин К. С., Дорофеева О. С. Сравнение компилятора и интерпретатора на примере языков программирования С и Python // Вестник Пензенского государственного университета. 2024. № 4. С. 4–8.

Современные компьютеры не могут воспринимать слова, написанные на языке, который будет понятен человеку. Они способны понимать только программы, написанные на двоичном коде (машинный код). Для этого и используются компиляторы, интерпретаторы.

Компилятор – это специальная программа, которая преобразует исходный код, написанный на языке программирования, в вид, который будет понятен компьютеру. Работа компилятора, как правило, состоит из нескольких этапов. Во-первых, проверяется структура всего исходного кода

для того, чтобы убедиться, что грамматические правила компилируемого языка не были нарушены. Далее происходит преобразование в некоторую промежуточную форму, которая уже затем будет передана для перевода в машинный код или на низкоуровневый язык.

Таким образом, результатом выполнения всех этих действий становится исполняемая программа. По такому принципу работает большинство современных высокоуровневых языков.

Интерпретатор – это программное средство, которое выполняет код, написанный на языке программирования высокого уровня, напрямую, без предварительного перевода в байт-код. Интерпретатор считывает и выполняет код построчно. Он переводит каждую строку в машинные инструкции непосредственно перед выполнением следующей строки, что облегчает выявление любых ошибок и отладку кода.

Концепция работы интерпретатора по своей сути схожа с компилятором (перевод языка, понятного человеку, в машинный код), но сильно различается в подходе к анализу программы. Вместо компиляции всего исходного кода перед выполнением интерпретатор переводит исходный код в машинный код во время выполнения.

Рассмотрим основные принципы трансляции на примерах языков программирования C и Python.

Программа на языке C проходит следующие этапы сборки:

1. Препроцессинг.
2. Компиляция.
3. Ассемблирование.
4. Линковка.

Препроцессор − это программа, которая обрабатывает исходный файл перед передачей в компилятор. Она обрабатывает различные директивы, например, #include, #define, заменяет эти директивы соответствующим содержимым. Препроцессор также делает возможной условную компиляцию (#ifdef, #endif), когда определенные блоки кода в программе могут игнорироваться компилятором. После препроцессинга получается модернизированный файл, в котором команды препроцессора уже обработаны. Такие файлы имеют расширение «.i». Этот файл участвует в следующем этапе – компиляции.

На следующем этапе компилятор сначала проверяет наличие синтаксических ошибок в коде, при обнаружении которых уведомляет об ошибке. Затем преобразует код в низкоуровневый код на ассемблере. Файл на выходе имеет расширение «.s».

На этапе ассемблирования полученный ассемблерный код переводится в машинный. После выполнения данного преобразования формируется файл с расширением «.o» или «.obj», в котором содержится двоичный код, полученный в ходе обработки ассемблерного файла.

В конце работы происходит линковка файлов. Линковка – это процесс компоновки различных частей кода вместе, в результате чего получается один исполняемый файл. Компоновщик объединяет объектные файлы, созданные ассемблером, и все необходимые библиотеки, в результате генерируя конечный исполняемый файл [1]. На рис. 1 приведен полный цикл сборки программы.

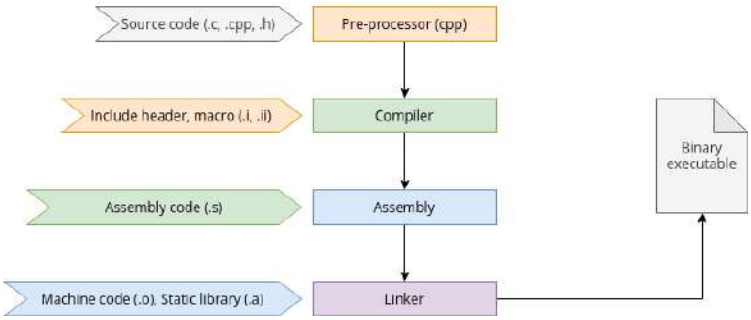


Рис. 1. Схема работы компилятора [1]

Программы на языке Python транслируются иным образом.

На первом этапе компилятор ищет ошибки в исходном коде, например, отсутствие отступов, неправильное название функции. При их обнаружении выбрасываются исключения.

Далее компилятор генерирует файлы с байт-кодом, которые имеют расширение «.pyc» [2]. На рис. 2 приведен байт-код программы, которая реализует функцию суммы двух целых чисел и выводит на экран сумму этих чисел.

```
  0           RESUME                   0

  1           LOAD_CONST               0 ('a')
              LOAD_NAME                0 (int)
              LOAD_CONST               1 ('b')
              LOAD_NAME                0 (int)
              LOAD_CONST               2 ('return')
              LOAD_NAME                0 (int)
              BUILD_TUPLE              6
              LOAD_CONST               3 (<code object sum at 0x000001E8436D25D0, file ".\article.py", line 1>)
              MAKE_FUNCTION
              SET_FUNCTION_ATTRIBUTE   4 (annotations)
              STORE_NAME               1 (sum)

  4           LOAD_NAME                2 (print)
              PUSH_NULL
              LOAD_NAME                1 (sum)
              PUSH_NULL
              LOAD_CONST               4 (1)
              LOAD_CONST               5 (2)
              CALL                     2
              CALL                     1
              POP_TOP
              RETURN_CONST             6 (None)

Disassembly of <code object sum at 0x000001E8436D25D0, file ".\article.py", line 1>:
  1           RESUME                   0

  2           LOAD_FAST_LOAD_FAST      1 (a, b)
              BINARY_OP                0 (+)
              RETURN_VALUE
```

Рис. 2. Пример байт-кода

Байт-код состоит из строк, которые образуют столбцы. Во втором столбце приведены коды операций, которые запускаются во время выполнения, например, POP_TOP, который удаляет элемент с вершины стека [3]. Из байт-кода также можно получить исходный код программы, для этого существуют различные утилиты, например, «python-uncompyle6», которая написана на Python.

Байт-код является промежуточным состоянием между исходным текстом программы и запущенным приложением. Он запускается виртуальной машиной Python, которая выполняет его построчно, руководствуясь кодом операции на текущей строке.

Код на Python может содержаться в нескольких файлах, использовать сторонние модули. В этом случае Python ищет модули в текущей директории, которую можно узнать с помощью метода os.getcwd(). Если нужный модуль не был найден, то поиск продолжается в директориях, которые перечислены в переменной среды PYTHONPATH. Для ускорения повторной загрузки сторонних модулей Python также может кешировать их байт-код [4].

Одним из главных преимуществ компилятора над интерпретатором является более высокая производительность результирующих программ. Для наглядного сравнения было проведено тестирование времени выполнения задачи вычисления суммы целых чисел от 1 до 999 999 999 на каждом из языков. Код на языке C:

```
#include <stdio.h>
#include <time.h>
void some_function() {
    long result = 0;
    for (long i = 1; i < 1000000000; i++) {
```

```
        result += i;
    }
}
int main() {
    clock_t start_time, end_time;
    double elapsed_time;
    start_time = clock();
    some_function();
    end_time = clock();
    elapsed_time = ((double) (end_time - start_time)) / CLOCKS_PER_SEC;
    printf("Time: %.6f sec\n", elapsed_time);
    return 0;
}
```

Выполнение функции заняло 3.154424 с.

Код на языке Python:

```
import time
def some_function():
  result = 0
  for i in range (1, 1000000000):
    result += i
  return result
start_time = time.time()
some_function()
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Time: {elapsed_time:.6f} sec")
```

Операция была завершена спустя 37.680010 с.

Из результатов теста видно, что время выполнения программы на C значительно меньше, чем на Python.

В заключение можно отметить, что компиляция и интерпретация – это два разных подхода к выполнению программ, каждый из которых обладает сильными и слабыми сторонами.

Компилятор обеспечивает более высокую скорость получаемых программ по сравнению с интерпретатором, так как переводит исходный код программы в машинный перед ее выполнением. Компиляторы применяют различные методы оптимизации на этапе компиляции, что снижает общее время работы. Однако недостатком использования данного подхода является невозможность быстрого исправления кода, так как право вносить изменения в программу имеется только при возвращении к исходному коду.

Интерпретируемые языки обрабатываются построчно интерпретатором, который считывает код и выводит результаты на экран по мере поступления. При этом код не может быть напрямую «прочитан» процессором. Он в любом случае переводится в понятный для компьютера код. Это делает интерпретируемые языки значительно медленнее на фоне компилируемых языков. Главным же преимуществом интерпретации является возможность более удобно и эффективно исправлять ошибки.

Следует отметить, что современные системы все чаще используют подходы, комбинирующие обе технологии, например, подход «компиляции на лету» (Just-in-Time compilation), который позволяет повысить эффективность выполнения кода, при этом сохраняя преимущества интерпретации. JIT-компиляторы динамически переводят части кода на машинный язык во время выполнения, тем самым обеспечивая баланс между скоростью и гибкостью [5].

Список литературы

1. A programmer's guide to GNU C Compiler // Opensource.com. URL: https://opensource.com/article/22/5/gnu-c-compiler (дата обращения: 01.10.2024).

2. An introduction to Python bytecodes // Opensource.com. URL: https:// opensource.com/article/18/4/introduction-python-bytecode (дата обращения: 05.10.2024).

3. Python Opcodes // SourceForge. URL: https://unpyc.sourceforge.net/Opcodes.html (дата обращения: 05.10.2024).

4. Modules // Python Docs. URL: https://docs.python.org/3/tutorial/modules.html (дата обращения: 08.10.2024).

5. Just-in-time compilation // Wikipedia. URL: https://en.wikipedia.org/wiki/Just-in-time_compilation (дата обращения: 10.10.2024).

Информация об авторах

Акчурин Али Радикович, студент, Пензенский государственный университет.

Симакин Константин Сергеевич, студент, Пензенский государственный университет.

Дорофеева Ольга Станиславовна, кандидат технических наук, доцент, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.021

АЛГОРИТМ РАССТАНОВКИ ПАЛАТОК МОБИЛЬНОГО ПУНКТА ВРЕМЕННОГО РАЗМЕЩЕНИЯ С ИСПОЛЬЗОВАНИЕМ СРЕДСТВ ГЕОИНФОРМАЦИОННОЙ СИСТЕМЫ «ЯНДЕКС.КАРТЫ»

И. О. Ащеулов[1], Д. В. Такташкин[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1]aigor200799@gmail.com
[2]taktashkin.dv@yandex.ru

Аннотация. При возникновении чрезвычайной ситуации одним из важнейших этапов ее нейтрализации является своевременная эвакуация оказавшегося на месте происшествия гражданского населения с последующим его временным переселением в безопасное место. Для этого создаются пункты временного размещения, в которых эвакуированные граждане смогут проживать на протяжении некоторого времени либо до появления возможности вернуться в свои дома после нейтрализации последствий возникшей чрезвычайной ситуации, либо до переселения в новое временное жилье. Одним из важнейших этапов создания пункта временного размещения является размещение объектов его инфраструктуры, которое должно быть произведено таким образом, чтоб учитывать все существующие нормативы. Представлен алгоритм, позволяющий разместить палатки, входящие в состав пункта временного размещения, на карте в соответствии с существующими нормативными документами.

Ключевые слова: чрезвычайные ситуации, жизнеобеспечение населения, пункты временного размещения, геоинформационные системы, размещение объектов на карте, «Яндекс.Карты»

Для цитирования: Ащеулов И. О., Такташкин Д. В. Алгоритм расстановки палаток мобильного пункта временного размещения с использованием средств геоинформационной системы «Яндекс.Карты» // Вестник Пензенского государственного университета. 2024. № 4. С. 9–13.

В современных условиях одну из главных угроз жизни и здоровью граждан, а также экономической и социальной стабильности страны в целом представляют различные чрезвычайные ситуации природного, техногенного и террористического характера. Для того, чтобы минимизировать ущерб, причиняемый возникающими чрезвычайными ситуациями, органам власти, в частности структурам Министерства по чрезвычайным ситуациям (МЧС), необходимо как можно более качественно и оперативно принимать меры по их нейтрализации.

Одним из важнейших этапов нейтрализации чрезвычайной ситуации является своевременная эвакуация оказавшегося на месте происшествия гражданского населения с последующим его временным переселением в безопасное место. Для этого группы сотрудников МЧС, посланные в район происшествия, создают для эвакуированных жителей пункты временного размещения (ПВР), где они смогут проживать на протяжении некоторого времени либо до появления возможности вернуться в свои дома после нейтрализации последствий возникшей чрезвычайной ситуации, либо до переселения в новое временное жилье. Однако в процессе создания пунктов временного размещения присутствует риск ошибок, вызванных человеческим фактором (например,

неудачного расположения объектов инфраструктуры ПВР или неверного расчета требуемого числа этих объектов), которые в перспективе могут затруднить проживание эвакуированных людей в ПВР или даже привести к причинению вреда их жизни и здоровью вследствие возникновения в этих пунктах новых чрезвычайных ситуаций. Все это делает весьма перспективной разработку автоматизированной системы поддержки принятия решений по развертыванию мобильных пунктов временного размещения при чрезвычайных ситуациях на основе технологий искусственного интеллекта.

Такой системе может понадобиться алгоритм размещения палаток, входящих в состав мобильного ПВР. Он должен основываться на существующих нормативных актах, регламентирующих работу пунктов временного размещения. Таковым являются «Методические рекомендации по организации первоочередного жизнеобеспечения населения в чрезвычайных ситуациях и работы пунктов временного размещения пострадавшего населения» [1]. В нем содержатся следующие требования к расположению палаток:

1. Мобильные ПВР размещаются на площади не более 400 м^2.

2. Палатки рекомендуется устанавливать группами общей вместимостью не более чем на 50 человек, но не более чем из 10 палаток. Внутри групп палатки следует устанавливать рядами. Расстояние между палатками в ряду должно быть не менее 3 м, а между рядами – не менее 5 м.

3. Разрывы между группами палаток должны быть не менее 15 м.

На основании вышеописанных условий можно составить формулы, на основе которых будут вычисляться координаты палаток размещаемого ПВР.

Пусть $TentWidth$ – ширина палатки, $TentLength$ – ее длина, $TentCapacity$ – вместимость, а $FullPeopleCount$ – число людей, размещаемых в ПВР.

Максимальное количество людей, которых можно разместить в одной группе палаток, определяется по следующей формуле, т.е. в одной группе может быть не более 10 палаток, в которых может быть размещено не более 50 человек.

$$GroupCapacity = \begin{cases} TentCapacity * 10, \text{если } TentCapacity < 5; \\ 50, \text{если } TentCapacity \geq 5. \end{cases}$$

Количество групп палаток вычисляется по формуле

$$GroupCount = \left\lceil \frac{FullPeopleCount}{GroupCapacity} \right\rceil.$$

Число людей, размещаемых в i-й группе палаток ПВР, определяется по формуле

$$PeopleCount_i = \begin{cases} GroupCapacity, \text{если } 1 \leq i \leq GroupCount - 1; \\ FullPeopleCount - GroupCapacity * (GroupCount - 1), \\ \text{если } i = GroupCount. \end{cases}$$

Пусть $TentGap = 3$ м – расстояние между палатками в одном ряду, $RowGap = 5$ м – расстояние между рядами палаток в группе, $GroupGap = 15$ м – расстояние между группами палаток в ПВР, $RowCount_i$ – число рядов в i-й группе палаток ПВР, $RowLength_i$ – максимальная длина ряда в i-й группе палаток ПВР.

Подберем оптимальную длину ряда для каждой из групп. Она должна быть такой, чтобы площадь, занимаемая группой, была наименьшей.

Число палаток в i-й группе определяется по формуле

$$TentCount_i = \left\lceil \frac{PeopleCount_i}{TentCapacity} \right\rceil.$$

Число рядов в i-й группе определяется по формуле

$$RowCount_i = \left\lceil \frac{TentCount_i}{RowLength_i} \right\rceil.$$

Длина группы вычисляется по формуле

$$GroupLength_i = RowLength_i * TentWidth + (RowLength_i - 1) * TentGap.$$

Ширина группы вычисляется по формуле

$$GroupWidth_i = RowCount_i * TentLength + (RowCount_i - 1) * RowGap.$$

Число палаток, недостающих в последнем ряду i-й группы до полного прямоугольника, задается формулой

$$MissingTents_i = RowCount_i * RowLength_i - TentCount_i.$$

Площадь, занимаемая *i*-й группой палаток, вычисляется по формуле

$$GroupArea_i = GroupLength_i * GroupWidth_i - TentLength \times \\ \times (MissingTents_i * TentWidth + (MissingTents_i - 1) * TentGap).$$

Таким образом, встает задача оптимизации:

$$GroupArea_i \to min \begin{cases} GroupArea_i < 400; \\ 1 \le RowLength_i \le TentCount_i. \end{cases}$$

После решения этой задачи и нахождения оптимального значения переменной $RowLength_i$ можно по перечисленным выше формулам вычислить остальные свойства i-й группы палаток.

Группы палаток будут размещены линиями, направленными перпендикулярно направлению рядов внутри групп. При этом максимальное число групп в этих линиях $GroupsInLine$ и число линий $GroupLinesCount$ должны быть подобраны так, чтобы форма ПВР была максимально близка к квадрату. Для этого воспользуемся формулами:

$$GroupsInLine = Round(\sqrt{GroupCount}),$$

где $Round(x)$ – функция округления *x* до ближайшего целого числа;

$$GroupLinesCount = \left\lceil \frac{GroupCount}{GroupsInLine} \right\rceil.$$

Пусть $x_{i,j,k}$ и $y_{i,j,k}$ – координата левого верхнего (на карте ПВР) угла его k-й палатки *j*-го ряда *i*-й группы (в метрах), где значения индексов i, j, k удовлетворяют условиям

$$1 \le i \le GroupCount;$$
$$1 \le j \le RowCount_i;$$
$$1 \le k \le RowLength_i, \text{если } j < RowCount_i;$$
$$1 \le k \le TentCount_i - (j-1) * RowLength_i, \text{если } j = RowCount_i.$$

Тогда координаты его правого нижнего угла $x^2_{i,j,k}$ и $y^2_{i,j,k}$ можно будет вычислить по формулам:

$$x^2_{i,j,k} = x_{i,j,k} + TentWidth,$$
$$y^2_{i,j,k} = y_{i,j,k} + TentLength.$$

Координаты различных палаток связаны следующими взаимоотношениями:

$$x_{i,j,k} = x_{i,j,k-1} + TentWidth + TentGap;$$
$$y_{i,j,k} = y_{i,j,k-1;}$$
$$x_{i,j,1} = x_{i,j-1,1};$$
$$y_{i,j,1} = y_{i,j-1,1} + TentLength + RowGap;$$
$$x_{i,1,1} = x_{i-1,1,1}, \text{если } i \bmod GroupsInLine \ne 1;$$
$$y_{i,1,1} = y_{i-1,1,1} + GroupWidth_{i-1} + GroupGap,$$

если $i \bmod GroupsInLine \neq 1$;

$$x_{i,1,1} = x_{i-GroupsInLine,1,1} + GroupLength_{i-GroupsInLine} + GroupGap,$$

если $i \bmod GroupsInLine = 1$;

$$y_{i,1,1} = y_{i-GroupsInLine,1,1}, \text{ если } i \bmod GroupsInLine = 1.$$

Таким образом, зная координаты самой первой палатки ПВР $x_{1,1,1}$ и $y_{1,1,1}$, можно будет легко найти координаты всех остальных палаток.

В качестве точки, от которой начинается подъезд к ПВР от ближайшей дороги, можно выбрать середину верхней границы лагеря. Тогда ее координаты можно задать по формулам:

$$x_{entr} = x_{1,1,1} + \frac{CampLength}{2},$$
$$y_{entr} = y_{1,1,1.}$$

Здесь $CampLength$ – длина верхней границы ПВР, вычисляемая по формуле

$$CampLength = GroupLength_0 * GroupLinesCount + \\ + GroupGap * (GroupLinesCount - 1).$$

Для перехода от условной системы координат внутри лагеря к стандартной системе географических координат на основе широты и долготы воспользуемся встроенными средствами используемой геоинформационной системы (ГИС) «Яндекс.Карты», позволяющими находить точку, удаленную от указанной на заданное расстояние в заданном направлении [2]. Направления определим следующим образом.

Пусть α – азимут направления подъезда к ПВР от ближайшей дороги (его также можно вычислить встроенными средствами ГИС [2, 3]). Примем его равным направлению вверх. Тогда направление вправо будет равно $\alpha + \frac{\pi}{2}$, вниз $-\alpha + \pi$, а влево $-\alpha + \frac{3\pi}{2}$. Полученные значения азимутов направлений будут переданы на вход геоинформационной системе, благодаря чему та сможет повернуть палатки ПВР именно под тем углом, под каким это будет наиболее оптимально.

Ниже представлены примеры работы предложенного алгоритма в различных условиях. На рис. 1 показан результат его применения для расположения на карте палаток ПВР на 500 человек (с использованием палаток на 24 человека длиной 8,15 м и шириной 5,3 м).

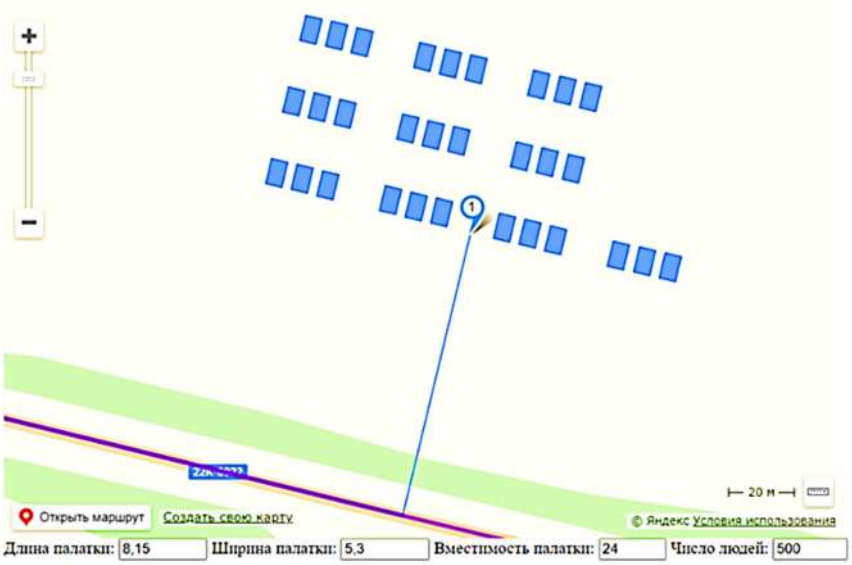


Рис. 1. Пример работы алгоритма для ПВР на 500 человек

Пример применения алгоритма для размещения вблизи водоема на карте палаток ПВР на 675 человек (с использованием палаток на 15 человек длиной 6,8 м и шириной 4,1 м) представлен на рис. 2.

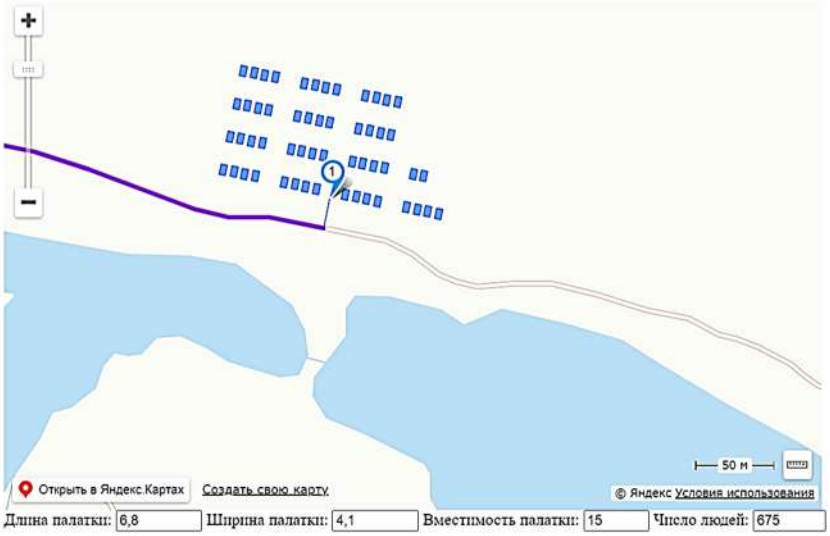


Рис. 2. Пример работы алгоритма для ПВР на 675 человек

Таким образом, разработанный алгоритм может быть успешно использован в автоматизированной системе поддержки принятия решений по развертыванию мобильных пунктов временного размещения при чрезвычайных ситуациях для решения задачи расположения палаток внутри ПВР.

Список литературы

1. Методические рекомендации по организации первоочередного жизнеобеспечения населения в чрезвычайных ситуациях и работы пунктов временного размещения пострадавшего населения. URL: https://mchs.gov.ru/dokumenty/2124 (дата обращения: 27.10.2024).

2. ICoordSystem. URL: https://yandex.ru/dev/jsapi-v2-1/doc/ru/v2-1/ref/reference/ICoordSystem (дата обращения: 27.10.2024).

3. Route. URL: https://yandex.ru/dev/jsapi-v2-1/doc/ru/v2-1/ref/reference/route (дата обращения: 27.10.2024).

Информация об авторах

Ащеулов Игорь Олегович, аспирант, Пензенский государственный университет.

Такташкин Денис Витальевич, кандидат технических наук, доцент, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.05

СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ СОРТИРОВКИ В JAVA: QUICKSORT, MERGESORT, HEAPSORT

А. А. Бауэр[1], А. Ю. Афонин[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1]artemiii1208@gmail.com
[2]afonin@pnzgu.ru

Аннотация. Проводится сравнительный анализ производительности трех широко используемых алгоритмов сортировки в Java: Quicksort, Mergesort и Heapsort. Исследование направлено на определение наиболее эффективного алгоритма для сортировки больших массивов данных (> n, v). Для анализа используются три типа тестовых наборов: случайный, отсортированный и обратно отсортированный.

Ключевые слова: алгоритм сортировки, Quicksort, Mergesort, Heapsort, Java, производительность, большой объем данных

Для цитирования: Бауэр А. А., Афонин А. Ю. Сравнительный анализ алгоритмов сортировки в Java: Quicksort, Mergesort, Heapsort // Вестник Пензенского государственного университета. 2024. № 4. С. 14–18.

Введение

Алгоритмы сортировки занимают важное место в области компьютерных наук и находят применение в самых различных сферах, таких как анализ данных, управление базами данных, искусственный интеллект и др. Правильный выбор алгоритма сортировки может существенно повысить производительность программ, особенно когда речь идет о больших объемах данных.

Основная часть

1. Реализация алгоритмов

В ходе исследования были разработаны три метода сортировки на языке Java. Коды реализации алгоритмов представлены на рис. 1−3.

```
// Quicksort
private static void quickSort(int[] array, int low, int high) {
    if (low < high) {
        int partitionIndex = partition(array, low, high);
        quickSort(array, low, high: partitionIndex - 1);
        quickSort(array, low: partitionIndex + 1, high);
    }
}

// Partition (для Quicksort)
private static int partition(int[] array, int low, int high) {
    int pivot = array[high];
    int i = (low - 1);
    for (int j = low; j < high; j++) {
        if (array[j] <= pivot) {
            i++;
            swap(array, i, j);
        }
    }
    swap(array, i: i + 1, high);
    return i + 1;
```

Рис. 1. Реализация Quicksort

Quicksort. Рекурсивный метод, при котором массив делится на две части, определяемые опорным (pivot) элементом. Механика работы данного алгоритма:

– выбирается опорный элемент;

– массив разбивается на три группы: элементы, которые меньше опорного, сам опорный и элементы, которые больше его;

– для каждой части рекурсивно вызывается quicksort [1].

```
// Heapsort
private static void heapSort(int[] array) {  1 usage
    int n = array.length;
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(array, n, i);
    }
    for (int i = n - 1; i >= 0; i--) {
        swap(array,  i: 0, i);
        heapify(array, i,  i: 0);
    }
}

// Heapify (для Heapsort)
private static void heapify(int[] array, int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && array[left] > array[largest]) {
        largest = left;
    }
    if (right < n && array[right] > array[largest]) {
        largest = right;
    }
    if (largest != i) {
        swap(array, i, largest);
        heapify(array, n, largest);
    }
}
```

Рис. 2. Реализация Heapsort

```
// Mergesort
private static void mergeSort(int[] array, int low, int high) {  3 usa
    if (low < high) {
        int mid = (low + high) / 2;
        mergeSort(array, low, mid);
        mergeSort(array,  low: mid + 1, high);
        merge(array, low, mid, high);
    }
}

// Merge (для Mergesort)
private static void merge(int[] array, int low, int mid, int high) {
    int n1 = mid - low + 1;
    int n2 = high - mid;
    int[] leftArray = new int[n1];
    int[] rightArray = new int[n2];
    for (int i = 0; i < n1; i++) {
        leftArray[i] = array[low + i];
    }
    for (int j = 0; j < n2; j++) {
        rightArray[j] = array[mid + 1 + j];
    }
    int i = 0, j = 0, k = low;
    while (i < n1 && j < n2) {
        if (leftArray[i] <= rightArray[j]) {
            array[k] = leftArray[i];
            i++;
        } else {
            array[k] = rightArray[j];
            j++;
        }
        k++;}
    while (i < n1) {
        array[k] = leftArray[i];
        i++;
        k++;}
    while (j < n2) {
        array[k] = rightArray[j];
        j++;
        k++;}
}
```

Рис. 3. Реализация Mergesort

Heapsort. Алгоритм сортировки, который основан на двоичной куче. Механика работы данного алгоритма:

– из исходного массива формируется двоичная куча;

– максимальный элемент (корень) удаляется и замещается последним элементом кучи;

– выполняется восстановление кучи до тех пор, пока не останется один элемент [2].

Mergesort. Метод, который делит массив на две половины, рекурсивно сортирует и сливает в отсортированный массив. Механика работы данного алгоритма:

– массив делится на две части;

– для каждой половины рекурсивно вызывается mergesort;

– отсортированные половины объединяются в один массив [1].

2. Создание тестовых наборов

Для тестирования алгоритмов были созданы три разных типа наборов данных:

– случайный набор: массив целых чисел в диапазоне от 0 до 1000, который сгенерирован с помощью класса Random;

– отсортированный набор: массив из 10 000 целых чисел, который отсортирован в порядке возрастания;

– обратно отсортированный набор: массив из 10 000 целых чисел, который отсортирован в порядке убывания [2].

Коды генерации наборов представлены на рис. 4–6.

```
// Генерация случайного набора
private static int[] generateRandomArray(int n) {
    Random random = new Random();
    int[] array = new int[n];
    for (int i = 0; i < n; i++) {
        array[i] = random.nextInt( bound: 1000);
    }
    return array;
}
```

Рис. 4. Генерация случайного набора

```
// Генерация отсортированного набора
private static int[] generateSortedArray(int n) {
    int[] array = new int[n];
    for (int i = 0; i < n; i++) {
        array[i] = i;
    }
    return array;
}
```

Рис. 5. Генерация отсортированного набора

```
// Генерация обратно отсортированного набора
private static int[] generateReverseSortedArray(int n) {
    int[] array = new int[n];
    for (int i = 0; i < n; i++) {
        array[i] = n - i - 1;
    }
    return array;
}
```

Рис. 6. Генерация обратно отсортированного набора

3. Проведение тестов

Для каждого алгоритма проводилось тестирование на каждом типе тестового набора. Время выполнения алгоритмов измерялось с помощью метода System.nanoTime() [3].

Код для тестирования позволяет измерить время выполнения каждого алгоритма сортировки на разных типах данных и получить результаты. Сам код представлен на рис. 7.

```
// Тестирование алгоритма
private static void testAlgorithm(String algorithmName, int[] array, String dataSetName) {
    long startTime = System.nanoTime();
    switch (algorithmName) {
        case "Quicksort":
            quickSort(array, low: 0, high: array.length - 1);
            break;
        case "Mergesort":
            mergeSort(array, low: 0, high: array.length - 1);
            break;
        case "Heapsort":
            heapSort(array);
            break;
    }
    long endTime = System.nanoTime();
    double duration = (endTime - startTime) / 1000000.0; // Время выполнения в миллисекундах
    System.out.println(algorithmName + " - " + dataSetName + ": " + duration + " мс");
}
```

Рис. 7. Код для тестирования алгоритмов сортировки

4. Результаты и их анализ

Результаты тестирования на различных наборах данных (рис. 8).

Случайный набор. Quicksort, как правило, показывает наилучшие результаты при работе со случайными данными.

Отсортированный набор. Quicksort демонстрирует замедление на отсортированных данных из-за того, что метод разбиения постоянно выбирает последний элемент в качестве опорного (pivot), что приводит к неэффективному разбиению. Mergesort и Heapsort показывают стабильные результаты [2].

Обратно отсортированный набор. В аналогичной ситуации Quicksort снова замедляется на обратно отсортированных данных, поскольку метод разбиения выбирает первый элемент как опорный (pivot), что также приводит к небалансу. Mergesort и Heapsort вновь показывают стабильную производительность [1].

Анализ результатов:

– Mergesort очень стабильный алгоритм. Он показал хорошие результаты на всех типах данных;

– Heapsort также показал достойные результаты для всех наборов, но он может быть не таким быстрым, как Mergesort, в некоторых ситуациях;

– Quicksort наиболее эффективен для случайных наборов, однако его эффективность значительно снижается на отсортированных и обратно отсортированных наборах.

```
Quicksort - Случайный набор: 3.2587 мс
Quicksort - Отсортированный набор: 68.5234 мс
Quicksort - Обратно отсортированный набор: 38.2199 мс


Mergesort - Случайный набор: 3.4659 мс
Mergesort - Отсортированный набор: 1.5476 мс
Mergesort - Обратно отсортированный набор: 1.4617 мс


Heapsort - Случайный набор: 3.1664 мс
Heapsort - Отсортированный набор: 2.5889 мс
Heapsort - Обратно отсортированный набор: 2.501 мс

Process finished with exit code 0
```

Рис. 8. Результаты работы программы

5. Замечания

В процессе тестирования были выявлены такие особенности:

1. Для крупных объемов данных (> n, v) и случайных наборов: алгоритм Quicksort может оказаться наиболее эффективным, если его реализовывать с учетом худших сценариев.

2. В случаях с большими объемами данных (> n, v) и частично отсортированными наборами алгоритм Mergesort демонстрирует наибольшую стабильность.

3. Когда важна стабильность в более сложных условиях, предпочтение следует отдать алгоритму Mergesort [4].

Заключение

Проведенный анализ способствует выявлению наилучшего алгоритма сортировки в зависимости от типа данных и конкретных задач. Среди всех известных алгоритмов Mergesort выделяется как наиболее стабильный и надежный, особенно при работе с большими массивами. В то же время алгоритм Quicksort может показывать лучшие результаты по скорости обработки данных на случайных наборах, однако его реализация требует особого внимания для того, чтобы избежать неблагоприятных ситуаций.

Список литературы

1. Knuth D. The Art of Computer Programming. Vol. 3: Sorting and Searching. Addison-Wesley, 1997. 780 p.

2. Sedgewick R. Heapsort: Algorithm, Implementation, and Analysis // Journal of algorithms. 1993. № 15. P. 76−100.

3. Самуйлов С. В. Алгоритмы и структуры обработки данных : учеб. пособие. Саратов : Вузовское образование, 2016. 132 с.

4. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. MIT Press, 2009. URL: openlibrary.org>books

Информация об авторах

Бауэр Артем Александрович, студент, Пензенский государственный университет.

Афонин Александр Юрьевич, кандидат технических наук, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.05

ТЕОРЕТИЧЕСКИЙ АНАЛИЗ И СРАВНЕНИЕ ПРАКТИЧЕСКОЙ СКОРОСТИ АЛГОРИТМОВ СОРТИРОВКИ МЕТОДОМ ШЕЛЛА

А. А. Бауэр[1], С. В. Самуйлов[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1]artemiii1208@gmail.com
[2]sws_p@mail.ru

Аннотация. Рассматривается классический алгоритм Шелла и его модификация с выбором интервала в заданном диапазоне. Проводится сравнение скорости работы этих алгоритмов на данных, в которых большие значения группируются в начале и конце списка. Показано, что такой алгоритм имеет ту же теоретическую вычислительную сложность, что и оригинал. В отдельных случаях модифицированный алгоритм показывает более высокую практическую скорость, но в серии испытаний классический метод Шелла работает быстрее.

Ключевые слова: сортировка, алгоритмы внутренней сортировки, сортировка Шелла, модифицированная сортировка Шелла, выбор шага

Для цитирования: Бауэр А. А., Самуйлов С. В. Теоретический анализ и сравнение практической скорости алгоритмов сортировки методом Шелла // Вестник Пензенского государственного университета. 2024. № 4. С. 19–21.

Сортировка списков, наряду с сортировкой массивов, является одной из важных задач в программировании. Правильный выбор структуры данных и алгоритма для конкретной задачи влияют как на результат сортировки, так и на ее эффективность, т.е. на скорость работы и требуемый объем оперативной памяти.

Скорость сортировки зависит не только от выбранного алгоритма, но и от природы сортируемых данных и способа их хранения:

– данные могут находиться в массиве или списке; одна и та же операция над массивом и списком выполняется с разной скоростью. Например, добавление элемента в массив требует сдвига в среднем половины его элементов, в то время как для вставки нового элемента в список нужно просто изменить указатели;

– данные могут быть уже отсортированы, отсортированы в обратном порядке, являться случайными с некоторым законом распределения;

– данные могут быть бесконечными (поступают на вход постоянно) или конечными, но очень большими.

В алгоритмах внутренней сортировки данные ограничены по объему, их можно загрузить в оперативную память и выполнять любые операции, а для хранения очень больших данных уже необходимы внешняя память и алгоритмы внешней сортировки, которые предполагают только последовательные операции [1].

Для сравнения алгоритмов по скорости работы используется математическая теоретическая оценка О-нотация. Эта оценка показывает, какое количество операций необходимо выполнить

в худшем, среднем и лучшем случае. Оценка показывает не точное число, а порядок, например $O(n^2)$ или $O(n^3)$, где n – длина сортируемых данных, т.е. такая оценка не может быть $O(n^2+1000)$ потому, что 1000 значительно меньше n^2 при росте n, но она может быть, например, $O(n^{7/6})$.

Сортировка вставками [2] применима и к спискам, и к массивам. В этой сортировке из сортируемых данных берется один элемент и помещается в правильную позицию среди ранее упорядоченных элементов:

– наилучший случай $O(n)$, когда начальные данные уже упорядочены, и мы просто копируем n элементов из данных в результат;

– наихудший случай $O(n^2)$, когда начальные данные отсортированы в обратной последовательности. Берем один элемент исходных данных и сравниваем с каждым из уже отсортированных. При этом каждый из n исходных элементов сравнивается с каждым из n уже отсортированных элементов;

– среднее время также составит $O(n^2)$, поскольку вероятность встретить большое или маленькое число в начале последовательности одинакова.

Сортировка Шелла [3] представляет собой оптимизированный алгоритм сортировки вставками. Цель этой оптимизации – уменьшить среднее время сортировки. У Шелла в первом проходе сравниваются не соседние элементы входной последовательности, а элементы на расстоянии d_1, во втором проходе – на расстоянии d_2, причем $d_2 < d_1$, и так далее до расстояния 1 (т.е. до сравнения соседних элементов).

Получается, что последний проход – это сортировка вставками. Например, $d_1 = 5$, $d_2 = 3$, $d_3 = 1$. Здесь первые два прохода упорядочивают «куски» входной последовательности, а последний – соединяет эти куски правильным образом.

Среднее время работы алгоритма Шелла зависит от выбора шагов d. Дональд Шелл предложил следующий алгоритм вычисления шагов: $d_1 = n/2$, $d_i = d_{i-1}/2$, $d_k = 1$. При этом наихудшая оценка алгоритма составит $O(n^2)$.

Роберт Седжвик [4] доказал, что для последовательности $d_i = 9 \cdot 2^i - 9 \cdot 2^{i/2} + 1$ (если i – четное) и $d_i = 8 \cdot 2^i - 6 \cdot 2^{(i+1)/2} + 1$ (i – нечетное) худшая оценка составит $O(n^{4/3})$.

Марцина Циура [5] опубликовал наилучшую эмпирическую (т.е. полученную на экспериментах) последовательность $d = \{1, 4, 10, 23, 57, 132, 301, 701, 1750\}$ для сортировки последовательности размером до 4000 элементов.

В этой работе проводится сравнение практических скоростей классической сортировки Шелла и сортировки, в которой параметр d принимает случайные значения. Для оценки скорости работы алгоритмов подсчитывается количество проходов по данным до достижения полного упорядочивания. Исходные данные сгенерированы случайным образом (согласно равномерному распределению) и имеют размер 4000 элементов. Были созданы две выборки с равномерным распределением и две выборки, в которых в начале находятся большие значения.

Результаты сравнения алгоритмов приведены на рис. 1, 2. Сначала выполнен эксперимент на одной выборке.

```
Сгенерировано 1 раз по 4 набора данных
Среднее количество операций при равномерном распределении данных
Классический Шелл: 27572
   Случайный Шелл: 22003
Среднее количество операций, если большие числа чаще встречаются в начале данных
Классический Шелл: 35580
   Случайный Шелл: 35026
```

Рис. 1. Сравнение классической и модифицированной сортировки

Однократный эксперимент продемонстрировал, что алгоритм сортировки Шелла со случайной последовательностью шагов d_i порой оказывается более эффективным, чем классический, как на вы-

борках с равномерным, так и с неравномерным распределением. Тем не менее, при проведении нескольких тестов именно классический алгоритм продемонстрировал наилучшие результаты.

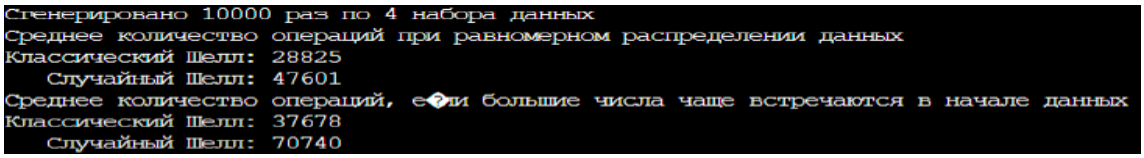


Рис. 2. Результат многократного сравнения

В дальнейшем алгоритмы анализировались несколько раз с использованием различных случайных значений d_i. Повторные испытания показывают, что сортировка Шелла, использующая случайный выбор шагов d_i, в среднем оказывается примерно в два раза менее эффективной по сравнению с классическим методом, при этом на равномерно распределенных данных случайный алгоритм показывает немного лучшие результаты, чем на неравномерных.

Это может объясняться тем, что генератор случайных чисел, применяемый для выбора d_i, в основном приводит к равномерному распределению.

В начале работы мы упоминали о наиболее эффективной эмпирической последовательности для данных, содержащих до 4000 элементов, которая превосходит классический метод Шелла. В проведенных нами экспериментах наша случайная последовательность не совпадала с этой рекомендованной последовательностью.

Из всего изложенного можно заключить, что классический алгоритм Шелла при сортировке до 4000 элементов продемонстрировал более чем двукратное преимущество в производительности по сравнению с алгоритмом Шелла, использующим генератор случайных чисел.

В то же время стоит предположить, что любое другое распределение, отличающееся от равномерного, возможно, сможет предоставить последовательность, приближенную к идеальной эмпирической. Если такое распределение будет обнаружено, можно ожидать улучшения работы классического алгоритма Шелла для наборов данных, превышающих 4000 элементов, для которых наилучшие эмпирические последовательности пока еще не опубликованы.

Список литературы

1. Самуйлов С. В., Самуйлова С. В. Структуры данных. Алгоритмы поиска и сортировки : учеб. пособие. М., 2024. 80 с.

2. Кнут Д. Э. Искусство программирования. Т. 3. Сортировка и поиск. Диалектика. 2019. 832 с. URL: litres.ru>Дональд Кнут

3. Shell D. L. A high-speed sorting procedure // Communications of the ACM. 1959. Vol. 2, Iss. 7. P. 30–32.

4. Incerpi J., Sedgewick R. Improved Upper Bounds for Shellsort // J. Computer and System Sciences. 1985. Vol. 31, № 2. P. 210–224.

5. Эмпирическая последовательность A102549. URL: https://oeis.org/A102549 (дата обращения: 27.10.2024).

Информация об авторах

Бауэр Артем Александрович, студент, Пензенский государственный университет.

Самуйлов Сергей Владимирович, кандидат технических наук, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.023

РАЗРАБОТКА АЛГОРИТМА ГЕНЕРАЦИИ ГРАФО-ХРОМАТИЧЕСКИХ КАРТ ДЛЯ РЕШЕНИЯ ЗАДАЧ КЛАССИФИКАЦИИ

А. С. Бождай[1], Л. Н. Горшенин[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1]bozhday@yandex.ru
[2]gorshenin.lev@gmail.com

Аннотация. В научных задачах особую актуальность имеет проблема классификации объектов с гетерогенным пространством информационных признаков. Представлена реализация алгоритма генерации графо-хроматических карт в рамках системы классификации объектов с гетерогенным пространством информационных признаков. Приведена UML-диаграмма активностей, описывающая данный алгоритм. Показан пример работы предложенного алгоритма, и сделаны выводы о потенциальных возможностях его применимости.

Ключевые слова: графо-хроматическая карта, генерация графо-хроматических карт, алгоритм, классификация, гетерогенное пространство признаков

Для цитирования: Бождай А. С., Горшенин Л. Н. Разработка алгоритма генерации графо-хроматических карт для решения задач классификации // Вестник Пензенского государственного университета. 2024. № 4. С. 22–26.

Введение

В научных исследованиях задача классификации [1] нередко осложняется тем, что классифицируемые объекты имеют гетерогенное пространство информационных признаков. Для решения таких задач был предложен оригинальный подход, описанный в [2].

Ключевая особенность данного подхода заключается в преобразовании гетерогенного пространства информационных признаков в унифицированное графическое представление – растровую графо-хроматическую карту (далее – ГХК). ГХК передается в качестве входных данных классификатору, который по ней определяет класс объекта. Изложенный подход позволяет классифицировать объекты с признаками самых разных типов (при условии существования алгоритма преобразования данных этого типа в графический формат), учитывать и настраивать важность признаков для классификации, а также производить классификацию по поднабору признаков – без перестроения или переобучения классификатора. Это значительно упрощает требования к самому классификатору, в качестве которого можно использовать одну из существующих архитектур сверточных нейронных сетей [3].

Проектирование алгоритма

С учетом сказанного можно определить следующие требования к алгоритму генерации ГХК:

1. На вход алгоритм должен принимать информацию об объекте и значения важности каждого признака для классификации.

2. На выходе алгоритм должен возвращать растровое изображение (ГХК), однозначно соответствующее объекту.

3. При генерации ГХК алгоритм должен учитывать важность признаков для классификации.

4. На сгенерированной ГХК должны быть отображены только признаки из поднабора признаков, по которому должна производиться классификация.

5. Каждому значению каждого признака на ГХК должна соответствовать своя метка (один или несколько графических примитивов). При этом преобразование значения признака в метку должно быть обратимым [2].

6. Алгоритм должен иметь возможность работы с различным числом признаков любых типов.

При проектировании было решено, что данные об объекте, подающиеся на вход алгоритма, будут иметь формат списка кортежей. Каждый кортеж будет содержать идентификатор признака объекта и значение этого признака. При этом по идентификатору признака можно будет получить полную информацию о признаке, включая: тип, путь к графическому примитиву, обозначающему признак, домен признака. Такой формат компактен и при этом гибок, так как отсутствует привязка к конкретным объектам, с конкретным набором признаков – список может содержать любое число признаков любых типов. При этом если признак имеет несколько значений, то можно просто добавить в список кортежи с идентификатором этого признака для каждого значения.

Так как в качестве классификатора будет использоваться сверточная нейронная сеть, цвет пустой ГХК должен быть черным (в формате RGB – 0,0,0) – в этом случае на вход нейросети будут поданы нули.

Семантика признаков будет отображаться на ГХК в виде графических примитивов, каждый из которых однозначно соответствует определенному признаку. Значения признаков было решено отображать на ГХК в виде меток (одного или нескольких графических примитивов). При этом одному значению должна соответствовать одна метка. Для удобства и гибкости реализации изначально можно формировать изображение в векторном формате (нанося на него последовательно векторные примитивы) и только после нанесения всей информации об объекте на ГХК преобразовывать изображение в растровый формат.

Значения численных признаков отображаются на ГХК в виде некоторого графического примитива, имеющего размер, соответствующий значению признака. При этом должно производиться масштабирование с учетом диапазона допустимых значений. В противном случае признак с большими возможными значениями будет в среднем отображаться на ГХК большим примитивом, чем признак, который может иметь только небольшие значения. Это может негативно сказаться на точности последующей классификации, так как область нанесения метки ограничена.

Значения категориальных признаков представляются в виде двух графических примитивов. Первый будет передавать семантику признака, второй – значение. Это обосновано тем, что значения категориальных признаков, в отличие от численных, не имеют отношения порядка [4]. Следовательно, отображения значений категориальных признаков на ГХК также не должны иметь отношения порядка.

Значения текстовых признаков кодируются в виде примитивного подобия QR-кода. Такой способ позволяет компактно кодировать текстовую информацию на ГХК. Строка в формате Unicode сериализуется в бинарную последовательность. Пиксели метки будут окрашиваться в соответствии с этой последовательностью (если бит 1, то цвет белый, иначе – черный). Графический примитив, выражающий семантику признака, будет помещаться в левом верхнем углу графического кода, не закрывая его. Метаданные о мультимедиапризнаках также удобно представлять в текстовом виде через сериализацию.

Графовые признаки предлагается кодировать в формате матрицы весов (в виде клетчатого поля). При этом каждая клетка будет соответствовать ребру, а ее цвет – весу.

Важность признаков можно передавать, задавая прозрачность примитивов соответствующего признака: чем меньше важность, тем менее заметен (вплоть до полной прозрачности) примитив на ГХК. Это упростит реализацию, так как, если классификация происходит по поднабору признаков и на ГХК должен быть нанесен только поднабор признаков, для неиспользуемых признаков можно просто установить значение важности в ноль, и они не будут отображены на ГХК. При этом алгоритму не потребуется иной информации, кроме значений важности признаков для классификации. Метки значений признаков позиционируются на ГХК построчно, одна за другой.

С учетом предложенных решений в обобщенном виде алгоритм выглядит следующим образом.

Входные данные: валидный список кортежей *<Идентификатор признака: Guid, значение признака: object>* и набор значений (0 до 255) важности каждого из признаков.

Выходные данные: растровое изображение ГХК.

1. Получение списка кортежей *<Идентификатор признака: Guid, значение признака: object>* от предыдущего блока;
2. Получение информации о важности каждого признака от блока управления. Важность признака для классификации передается через значение от 0 до 255;
3. Инициализация пустой ГХК.
4. Если список признаков пуст, то переход к шагу 13;
5. Выборка очередного элемента списка (кортежа *<идентификатор признака, значение признака>*);
6. Если важность признака с таким идентификатором равна нулю, то переход к шагу 4;
7. Получение информации о признаке по его идентификатору от менеджера признаков, в том числе: тип признака, путь к примитиву и параметры, зависящие от типа признака;
8. Подсчет позиции очередной метки;
9. Выбор стратегии нанесения информации о признаке и его значении на ГХК в зависимости от типа признака;
10. Нанесение метки значения признака с помощью выбранной стратегии на подсчитанную позицию ГХК с учетом значения признака;
11. Настройка прозрачности метки, в зависимости от важности;
12. Переход к шагу 4;
13. Преобразование ГХК в растровый формат.

Данному алгоритму можно поставить в соответствие UML-диаграмму активностей [5] для алгоритма генерации ГХК (рис. 1). Блок управления (ParametersManager) отвечает за хранение и передачу настроек, влияющих на генерацию и классификацию. Блок генерации ГХК (GCMGeneratorUnit) отвечает за процесс генерации ГХК в целом. Построитель ГХК (GCMBuilder) формирует растр ГХК. Менеджер признаков (FeatureManager) отвечает за информацию о признаках, которые генератор может отобразить на ГХК.

Реализация алгоритма

Для реализации системы в целом и алгоритма генерации ГХК, в частности, было решено использовать язык C#. Это объектно-ориентированный язык, обладающий широкими возможностями и имеющий множество доступных пакетов, в частности ML.Net, пакет, который предназначен для машинного обучения. Реализация выполнялась в Visual Studio 2022 Community Edition.

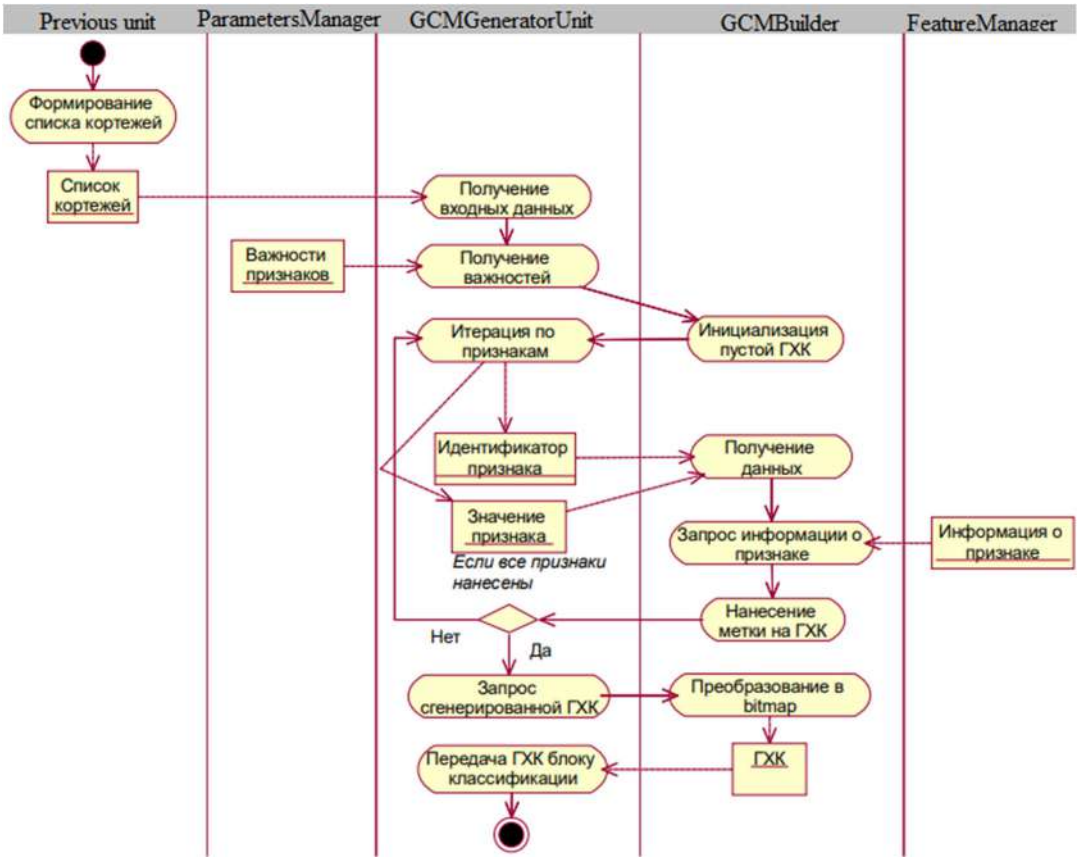


Рис. 1. Диаграмма генерации ГХК

В качестве исходных данных для демонстрации примера генерации ГХК используются данные о популярных именах с официального ресурса Data.gov [6]. Данные были взяты в формате csv файла. Файл содержал пять столбцов: год (численный), пол (категориальный), этнос (текстовый), имя (текстовый), количество данных имен (численный). На рис. 2 приведен фрагмент ГХК, сгенерированной по следующим данным (2011, FEMALE, HISPANIC, GERALDINE, 13). Для демонстрации важности признаков были установлены следующим образом: 128, 64, 255, 32, 255.

Рис. 2. Демонстрационный пример фрагмента ГХК

Заключение

В статье представлен алгоритм генерации ГХК в обобщенном виде. Его основным достоинством является возможность использования для автоматической классификации сложных объектов с гетерогенным набором признаков переменного размера. Унификация применения алгоритма

достигается за счет растровой графической формы ГКХ, что позволяет использовать стандартные нейросетевые модели для классификации растровой графики.

Список литературы

1. Jason Lim. What is data classification. URL: https://www.alation.com (дата обращения: 12.04.2024).

2. Бождай А. С., Горшенин Л. Н. Обзор и анализ подходов к классификации объектов с гетерогенным набором информационных признаков // Известия высших учебных заведений. Поволжский регион. Технические науки. 2024. № 2. С. 47–57. doi: 10.21685/2072-3059-2024-2-3

3. Waseem Rawat, Zenghui Wang. Deep convolutional neural networks for image classification: A comprehensive review // Neural Computation. 2017. № 29. P. 2352–2449.

4. Отношение порядка // Большая Российская Энциклопедия. URL: https://bigenc.ru (дата обращения: 28.10.2024).

5. Activity Diagrams – Unified Modeling Language (UML) // GeeksForGeeks. URL: https://www. geeksforgeeks.org (дата обращения: 25.10.2024).

6. The Home of the U. S. Government's Open Data. URL: https://data.gov (дата обращения: 16.10.2024).

Информация об авторах

Бождай Александр Сергеевич, доктор технических наук, профессор, профессор кафедры «Системы автоматизированного проектирования», Пензенский государственный университет.

Горшенин Лев Николаевич, аспирант, Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.6

МНОЖЕСТВЕННЫЙ БИНАРНЫЙ ДИСКРИМИНАНТНЫЙ АНАЛИЗ БИНАРНЫХ ДАННЫХ

П. М. Винник[1], М. В. Кобелев[2], М. С. Краенков[3]

[1,2,3]Балтийский государственный технический университет «ВОЕНМЕХ» им. Д. Ф. Устинова, Санкт-Петербург, Россия

[1] vinnik.pm@voenmeh.ru
[2] michalich1300@mail.ru
[3] kraenkovms@mail.ru

Аннотация. Рассматривается задача создания максимально быстрого и точного метода выявления принадлежности того или иного объекта конкретному типу на основе совокупности признаков, принимающих одно из двух значений – 0 или 1. Решение данной задачи непосредственно связано с вопросом обработки большого количества статистических данных, а также организации дальнейшего быстрого и простого доступа к результатам без повторного анализа. Представлен способ обработки данного типа входных данных.

Ключевые слова: дискриминантный анализ, бинарные данные, метод перебора, метод главных компонент, статистика

Для цитирования: Винник П. М., Кобелев М. В., Краенков М. С. Множественный бинарный дискриминантный анализ бинарных данных // Вестник Пензенского государственного университета. 2024. № 4. С. 27–30.

Введение

В настоящее время практически во всех сферах социально-экономических направлений приходится иметь дело с большим объемом данных, полученных в результате статистического наблюдения. Ввиду их объема (например, в [1] упомянут анализ 116 наблюдений, каждое из которых имело по 54 676 признаков) приходится каким-либо образом увеличить простоту их восприятия, доступа к ним, с целью понижения затрат ресурсов как человеческих, так и машинно-вычислительных.

Но практически невозможно сразу сказать, каким образом можно систематизировать исходные данные и быстро получать доступ к нужной информации, поэтому приходится разрабатывать различные алгоритмы обработки данных.

В данной статье рассмотрен множественный дискриминантный анализ такого набора входных данных, значение каждого из признаков которых является бинарным.

Постановка задачи

Пусть существует некоторая таблица, заданной размерности, столбцы которой указывают на признаки, а строки же – на конкретные наблюдения, в дальнейшем будем называть их образцами. Все образцы делятся на «типы», включающие в себя от одного до некоторого количества образцов.

Задача состоит в быстром предсказании, к какому именно типу относится выбранный образец только на основе признаков, задающих его.

В отличие от стандартных задач дискриминации в этой задаче не ставится задача определения типа, к которому принадлежит новый, не входящий в совокупность образец.

С целью применения вычислительных методов изначальная таблица представляется в виде матрицы $A = \begin{pmatrix} t_{i_1} & a_{11} & \dots & a_{1n} \\ t_{i_2} & a_{21} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ t_{i_m} & a_{m1} & \dots & a_{mn} \end{pmatrix}$ размерностью $m \times (n+1)$, где m – количество образцов, n – количество различных признаков, t_{i_j} – тип образца. В свою очередь, m образцов делятся на L типов, количество которых может варьироваться от $L = 2$ до $L = m$.

Метод главных компонент

В ходе решения задачи была предпринята попытка использования метода главных компонент (МГК) в одной из его вариаций [2, 3]. Для реализации использовался пакет MATLAB.

Суть метода заключается в понижении размерности входных данных с минимальной их потерей. С его помощью, действительно, получалось определить принадлежность образца типу. Но, во-первых, МГК базируется на весьма сложных вычислительных процедурах поиска собственных чисел и векторов матрицы крайне большой размерности, а, во-вторых, наличие достаточно большого количества типов и необходимость их полного разделения приводит к необходимости рассмотрения проекций множества многомерных точек на различные координатные плоскости, что малонаглядно. Ввиду указанных трудностей от этого метода пришлось отказаться и отдать предпочтение более универсальному методу, который помог бы в решении задачи и избежал недостатков метода главных компонент.

Можно предположить, что неудача с МГК обусловлена самим бинарным характером данных: сам МГК работает с количественными значениями признаков, следовательно, не учитывает их бинарный характер.

Множественный метод бинарной дискриминации бинарных данных

Идея метода состоит в том, что бинарность значений признаков позволяет по каждому образцу найти число T, двоичным разложением которого является набор значений признаков данного образца. Например, строкой значений признаков образца является $a_{j1}, K, \dots, a_{jn}$. Тогда $T = a_{j1} + 2a_{j2} + K \;\; 2^n a_{jn}$. Если образцам разных типов соответствуют одинаковые числа, то эти типы неразделимы. Полное разделение типов означает, что каждому типу соответствует свое число. Очевидно, в идеале нужно искать такой набор i_1, K, i_s признаков, для которых получаемые по бинарному набору $\mathrm{a}_{\mathrm{ji}_1}, \dots, \mathrm{a}_{\mathrm{ji}_s}$ числа T для всех типов различны, причем такой (оптимальный) набор может содержать существенно меньше признаков, чем всего имеется в наличии. Суть метода заключается в снижении количества признаков до минимального, с помощью которого можно было бы однозначно определять принадлежность образца, путем отсеивания малозначимых или бесполезных для разделения признаков. Одним из возможных решений является перебор всевозможных наборов признаков и выделение из них наборов, разделяющих все типы (или большинство типов) и включающих в себя наименьшее возможное количество признаков.

Так как сложность алгоритмов перебора – $O(n!)$, время их реализации велико. Однако процедуру перебора нужно проводить лишь единожды для конкретного набора данных, а затем лишь

оперировать с признаками оптимального набора, что очень сильно упрощает работу с образцами в дальнейшем. В целях сокращения времени перебора было использовано несколько способов, с помощью которых удалось сократить время поиска решений. Для ускорения работы программы, реализующей указанный перебор наборов, было проведено ее распараллеливание [4]. Оно сократило время работы программы примерно в 20 раз. Введение побитовых сравнений [5] позволило еще более ускорить перебор.

В результате выполнения перебора получаем сокращение количества признаков до некоторого неизвестного заранее значения. Так как это минимально возможное необходимое количество данных, то работа алгоритма заканчивается на данном этапе.

Представление каждого типа в виде числа, получаемого из бинарных значений отобранных признаков, существенно упрощает работу с данными для пользователя и позволяет ее автоматизировать.

Практическое применение

Таблицы бинарных данных могут возникать в огромном количестве областей, где происходит работа с человеком. Дискриминантный анализ таких таблиц может быть применен в любой сфере с качественным характером данных: социология, экономика, медицина. Такой анализ позволяет снизить время поиска информации, а следовательно, и увеличить продуктивность работы. Еще одним немаловажным достоинством этого анализа является способность к прогнозированию исходов на основе меньшего количества признаков.

Заключение

Все статистические данные изначально получаются опытным путем, следовательно, ненулевой является вероятность ошибки при получении значения того или иного признака. Этот вопрос планируется изучить в дальнейшем с применением того же дискриминантного анализа. Также стоит вопрос в создании такого метода поиска оптимального набора, который сможет работать без полного перебора всех возможных наборов признаков. Планируется также оптимизировать распараллеливание программ, реализующих указанные методы.

Список литературы

1. Бартош М. С., Масич И. С. Прогнозирование событий на основе объектов с большим количеством признаков // Решетневские чтения : материалы XXVII Междунар. науч.-практ. конф., посвящ. памяти генерального конструктора ракетно-космических систем, академика М. Ф. Решетнева. Красноярск, 2023. С. 47–49.

2. Айвазян А. С., Бухштабер В. М., Енюков И. С., Мешалкин Л. Д. Прикладная статистика: Классификация и снижение размерности. М. : Финансы и статистика, 1989. 609 с.

3. Эсбенсен К. Анализ многомерных данных. Избранные главы. Черноголовка: Изд-во ИПХФ РАН, 2005. 160 с.

4. Параллелизм в алгоритмах – выявление и рациональное его использование. Возможности компьютерного моделирования. URL: https://habr.com/ru

5. Степанов А. Н. Архитектура вычислительных систем и компьютерных сетей. СПб. : Питер, 2007. 509 с.

Информация об авторах

Винник Петр Михайлович, доктор технических наук, доцент, заведующий кафедры «Высшая математика», Балтийский государственный технический университет «ВОЕНМЕХ» им. Д. Ф. Устинова.

Кобелев Михаил Владимирович, студент, Балтийский государственный технический университет «ВОЕНМЕХ» им. Д. Ф. Устинова.

Краенков Марк Станиславович, студент, Балтийский государственный технический университет «ВОЕНМЕХ» им. Д. Ф. Устинова.

Авторы заявляют об отсутствии конфликта интересов.

УДК 378:004

АРХИТЕКТУРА ВЕБ-ПРИЛОЖЕНИЯ ДЛЯ ОЦЕНКИ ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ

И. В. Десятов[1], А. А. Свечников[2], А. В. Кузнецов[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1] ivandesyatov3@gmail.com
[2] kdsa3102@gmail.com
[3] vt@pnzgu.ru

Аннотация. Описывается архитектура системы для анализа веб-интерфейсов с использованием нейросети. Рассматриваются основные принципы построения системы, использующей искусственный интеллект, и функционал ее компонентов.

Ключевые слова: веб-интерфейс, анализ сайтов, распознавание элементов, пользовательский опыт, микросервисная архитектура, мобильные и десктопные приложения, нейросети, искусственный интеллект, высоконагруженные системы

Для цитирования: Десятов И. В., Свечников А. А., Кузнецов А. В. Архитектура веб-приложения для оценки пользовательских интерфейсов // Вестник Пензенского государственного университета. 2024. № 4. С. 31–34.

Искусственный интеллект (ИИ) все основательнее проникает в разнообразные сферы нашей жизни, становясь необходимой составляющей нынешних технологий. Благодаря ИИ получается существенно увеличить эффективность большого количества процессов, автоматизируя проблемы, какие прежде требовали существенных временных и человеческих ресурсов. Один из основных компонентов эффективной реализации ИИ – тщательно спроектированная структура системы, обеспечивающая ее бесперебойную работу, а кроме того, результативное взаимодействие между ее раздельными элементами.

Веб-приложения, предназначенные для оценки качества пользовательских интерфейсов разнообразных веб-сайтов, а кроме того, для распознавания и анализа компонентов их интерфейсов, строятся в основе комплексной архитектуры, включающей ряд основных частей. Каждая из этих частей играет немаловажную роль в общей работе системы, снабжая ее работоспособность, эффективность, а также безопасность. В состав системы входят следующие ключевые элементы:

1. Брокер сообщений выполняет функцию посредника между различными микросервисами системы, снабжая прочную и эффективную передачу информации. Он отвечает за организацию обмена данными, гарантируя, что все сообщения доставляются адресату в нужном порядке, а также вместе с требуемой скоростью. В высоконагруженных системах, где значимы скорость и безопасность передачи информации, брокер сообщений становится критически значимым компонентом, предотвращая вероятные перебои, а также обеспечивая целостность передаваемой информации.

2. Серверная часть системы считается ее центральным узлом, отвечающим за обработку входящих запросов от юзеров. Именно она обрабатывает сведения, прибывающие с клиентской части, а также взаимодействует вместе с остальными элементами системы, такими как база данных и нейронная сеть. Серверная часть несет ответственность за осуществление логики приложения, а кроме того, за интеграцию с внешними сервисами и API. Помимо этого, именно она управляет распределением нагрузки между разными микросервисами, что в особенности немаловажно для предоставления устойчивой работы в условиях больших нагрузок [1].

3. Пользовательский интерфейс (UI) является тем компонентом системы, вместе с которым напрямую взаимодействует юзер. От качества, а также удобства UI зависит пользовательский опыт, а кроме того, результативность работы вместе с приложением. В случае веб-приложений, оценивающих прочие веб-интерфейсы, UI обязан быть подсознательно ясным, а также эластичным, предоставляя юзеру доступ к абсолютно всем функциям системы, таким как отправление запросов в анализ интерфейса, просмотр итогов, а также регулирование настройками. Высококачественно разработанный пользовательский интерфейс значительно упрощает работу с системой, делая ее легкодоступной для обширного круга юзеров.

4. Файловое хранилище применяется с целью сохранения разных видов файлов, в том числе рисунков веб-интерфейсов, какие необходимо проанализировать. Оно обязано обеспечивать безопасное сохранение информации, стремительный доступ к ней, а также защиту с несанкционированного доступа. Эффективное управление файлами в хранилище имеет главное значение для обеспечения бесперебойной работы системы, так как от этого зависят темп, а также качество анализа информации. Немаловажно выделить, что файловое хранилище обязано сохранять масштабируемость, для того чтобы преодолевать увеличение объема информации по мере увеличения количества юзеров системы.

5. База данных считается главным хранилищем высокоструктурированной информации, нужной для работы системы. Именно она сохраняет сведения о юзерах, результаты анализа интерфейсов, данные о файлах, а также иных составляющих, значимых для функционирования системы. Инновационные базы данных гарантируют значительную эффективность, моментальный доступ к информации, а также возможность эффективной обработки. В системе, оценивающей веб-интерфейсы, база данных обязана сохранять крупные размеры данных, а также гарантировать их целостность, то, что дает возможность системе трудиться постоянно, а также надежно.

6. Нейронная сеть представляет собою главной элемент системы, осуществляющий функции анализа, а также распознавания компонентов веб-интерфейсов. Благодаря способностям машинного обучения, а также глубокого обучения нейронная сеть может подвергать обработке трудные проблемы, такие как распознавание компонентов интерфейса, анализ их удобства, а также предсказание взаимодействия юзера вместе с интерфейсом. Применение нейронных сетей дает возможность автоматизировать процесс анализа, существенно сократив время и действия, нужные для выполнения данной проблемы вручную [2]. Помимо этого, нейронная сеть может учиться на основе новейших сведений, что гарантирует ее приспособление к меняющимся условиям и потребностям юзеров.

Система может быть показана в виде схемы (рис. 1), объединяющей базу данных (БД), искусственный интеллект (ИИ) в виде нейронной сети, файловое хранилище и брокер сообщений для эффективного управления информацией и коммуникациями. Такая структура может быть реализована на основе микросервисов, что дает возможность достичь высокой гибкости, масштабируемости и надежности системы. Микросервисная архитектура делает систему легко адаптируемой к изменениям, а также дает возможность ей стремительно реагировать на изменения в потребностях пользователей или на технические изменения.

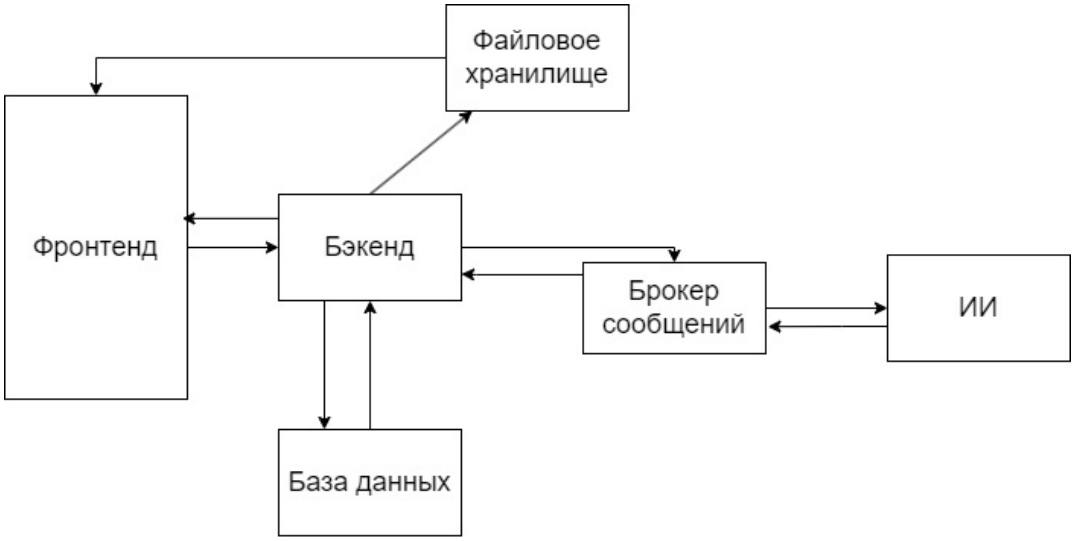


Рис. 1. Структура системы

Рассмотрим работу этой архитектуры. Пользователь отправляет с сайта запросы, которые обрабатываются на серверной части. Запрос может содержать изображение веб-интерфейса для анализа. При отправке запроса на анализ фотографии система сохраняет файл в хранилище, а после через брокера сообщений отравляет запрос в нейронную сеть для анализа.

База данных используется для хранения структурированной информации, такой как данные о пользователях, результаты обработки фотографии, информация о файлах и т.д. Благодаря БД допустимо продуктивно организовывать, а также подвергать обработке крупные объемы информации, гарантировать моментальный доступ к данным, а также обеспечивать целостность информации [1].

Файловое хранилище используется для хранения различных файлов, таких как изображения. Благодаря файловому хранилищу возможно эффективно управлять и хранить медиаконтент, обеспечивать доступ к нему и его безопасность [3].

Брокер сообщений в системе используется для обмена сообщениями между различными микросервисами системы, обеспечивая межкомпонентную коммуникацию. Благодаря брокеру сообщений можно обеспечить надежную и быструю передачу информации между компонентами системы, обеспечить целостность сообщений и обработку их в нужном порядке.

Заключение

Внедрение и совершенствование искусственного интеллекта имеют большой потенциал в разных сферах нашей жизни. Необходимо уделить внимание не только разработке алгоритмов и моделей, но и созданию эффективной и надежной структуры системы [4]. Предложенная архитектура обеспечивает высокую гибкость, масштабируемость и производительность системы, позволяя эффективно решать сложные задачи и обеспечивать надежную работу приложения [1].

Список литературы

1. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. Sprint Book, 2024. 638 с. URL: github.com>

2. Иванов И. И., Сидоров В. П. Машинное обучение для распознавания интерфейсов // Программная инженерия. 2022. № 4. С. 45–53.

3. Кузнецова Л. В., Бондаренко Н. И. Оценка удобства использования мобильных интерфейсов // UX Design. 2023. № 6. С. 78–85.

4. Траск Э. Грокаем глубокое обучение. Питер, 2021. 603 с. URL : habr.com>ru/companies/piter

Информация об авторах

Десятов Иван Вячеславович, студент, Пензенский государственный университет.

Свечников Артём Александрович, студент, Пензенский государственный университет.

Кузнецов Алексей Валерьевич, доцент кафедры «Вычислительная техника», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 378:004

ИНСТРУМЕНТАРИЙ ОЦЕНКИ САЙТОВ И РАСПОЗНАВАНИЕ ЭЛЕМЕНТОВ ВЕБ-ИНТЕРФЕЙСА

И. В. Десятов[1], А. А. Свечников[2], А. А. Сорокин[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1] ivandesyatov3@gmail.com
[2] kdsa3102@gmail.com
[3] vt@pnzgu.ru

Аннотация. Представлено описание веб-сайта с целью оценки качества сайтов, а также распознавания компонентов пользовательских интерфейсов. Рассматриваются ключевые проблемы, решаемые нашим вебсайтом, в том числе исследование структуры интерфейсов десктопных и мобильных приложений, оценку их пользовательского опыта, а также комфорт использования. Показана общая модель работы системы распознавания интерфейсов, описаны ее связь вместе с пользователем и возможные сценарии использования.

Ключевые слова: веб-интерфейс, оценка сайтов, распознавание элементов, пользовательский опыт, десктопные и мобильные приложения

Для цитирования: Десятов И. В., Свечников А. А., Сорокин А. А. Инструментарий оценки сайтов и распознавание элементов веб-интерфейса // Вестник Пензенского государственного университета. 2024. № 4. С. 35–38.

Введение

Современные веб-сайты и мобильные приложения играют важнейшую роль в повседневной жизни пользователей [1]. В условиях жесткой конкуренции на цифровом рынке успех того или иного продукта напрямую зависит от качества его интерфейса. Для создания удобных, функциональных и интуитивно понятных интерфейсов требуются тщательная оценка и анализ их элементов. Ручная проверка и тестирование интерфейсов может быть трудоемким и длительным процессом, особенно в условиях стремительно меняющихся стандартов и пользовательских предпочтений.

Для автоматизированной оценки веб-интерфейсов предлагается создать специализированный сайт. Его основная задача – помочь разработчикам, дизайнерам и аналитикам выявлять проблемные области в пользовательских интерфейсах. В данной статье мы рассмотрим, как такой инструмент может быть полезен в процессе разработки.

Одной из ключевых функций сайта для оценки других сайтов является анализ пользовательского опыта (UX). Пользовательский опыт – это комплекс ощущений, которые пользователь получает при взаимодействии с интерфейсом [2]. Важность UX сложно переоценить, так как он напрямую влияет на лояльность пользователей, их удовлетворенность и вероятность повторного использования ресурса.

Сайт проводит автоматизированный анализ различных аспектов UX, таких как:

1) навигация: оценивается удобство перемещения по сайту, простота доступа к основным разделам и возможность быстрого нахождения нужной информации;

2) интерактивность: анализируется, насколько удобно пользователю взаимодействовать с элементами интерфейса, такими как кнопки, формы и меню;

3) скорость и производительность: измеряется скорость загрузки страниц и взаимодействия с ними, что является важным фактором для удержания пользователей.

Распознавание элементов интерфейса

Сайт для оценки качества веб-интерфейсов также включает в себя функцию распознавания элементов интерфейса (рис. 1). Это позволяет автоматически идентифицировать и классифицировать такие элементы, как кнопки, текстовые поля, изображения, ссылки и т.д. [3].

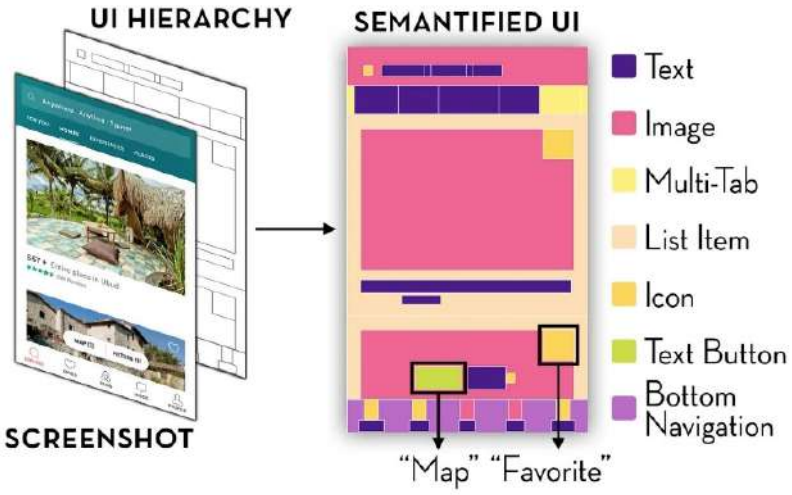


Рис. 1. Процесс распознавания и классификации элементов [4]

Рисунок подчеркивает, как технологии анализа интерфейса могут преобразовывать сложную структуру веб-страницы в упрощенную модель, позволяющую эффективно оценивать и оптимизировать элементы интерфейса. Такая семантическая карта помогает разработчикам и дизайнерам быстро идентифицировать и анализировать ключевые компоненты интерфейса для улучшения пользовательского опыта (UX).

Распознавание элементов интерфейса необходимо для следующих целей:

1) оценка доступности: автоматический анализ элементов на соответствие стандартам доступности, таким как правильное использование цветов, контраста и шрифтов;

2) проверка соответствия стандартам: распознанные элементы сравниваются с промышленными стандартами и рекомендациями по дизайну, что позволяет выявлять отклонения и предлагать их исправления;

3) анализ мобильной адаптации: оценка того, насколько элементы интерфейса оптимизированы для использования на мобильных устройствах (включая размер и размещение элементов).

Адаптация для мобильных устройств

С повышением количества пользователей, предпочитающих мобильные устройства с целью ежедневного доступа к Интернету, адаптирование веб-интерфейсов для этих устройств стало одной из наиболее важных проблем для разработчиков, а также дизайнеров. Качество взаимодей-

ствия юзера вместе с интерфейсом в мобильных платформах непосредственно воздействует на их восприятие веб-сайта, что устанавливает уровень удовлетворенности, а также преданности аудитории [2]. В обстоятельствах увеличения подвижного трафика предоставление полного приспособления веб-страниц служит важным обстоятельством эффективного функционирования каждого онлайн-ресурса.

Инновационные системы для оценки качества интерфейсов дают автоматизированные инструменты для анализа адаптивности веб-страниц. Данные системы гарантируют основательную проверку основных характеристик, которые оказывают большое влияние на удобство применения веб-сайта в мобильных устройствах:

1) *отзывчивость дизайна*.

Отзывчивый дизайн предполагает умение интерфейса автоматом приспосабливаться под разнообразные размеры экранов, а также ориентацию устройства (портретную либо ландшафтную). Это в особенности немаловажно для создания универсального опыта, что сохраняется вне зависимости от вида устройства, будь то телефон, планшет либо компьютер [1]. Автоматизированный анализ проводит проверку, в какой степени правильно веб-сайт отражается на разных экранах, оценивая компоненты, которые обязаны масштабироваться либо менять собственное положение, для того чтобы гарантировать наибольшую читабельность, а также удобство навигации. В отсутствие соответственного уровня отзывчивости юзер способен встретиться с такими трудностями, как неправильно воспроизводимый текст, частично скрытые рисунки, а также некомфортная навигация;

2) *удобство сенсорного управления*.

Взаимодействие с мобильными устройствами в основном совершается через сенсорные экраны, что требует особого интереса к исследованию сенсорных зон, а также жестов. Удобство сенсорного управления расценивается по нескольким аспектам, в том числе размер, а также расположение нажимных зон (к примеру, кнопок), какие обязаны являться довольно большими, а также удобно расположенными с целью нажатия пальцем. Помимо этого, анализируется скорость отклика интерфейса на действия юзера, то, что влияет на общую плавность, а также отзывчивость взаимодействия. Продолжительное ожидание отклика либо некомфортные элементы управления имеют все шансы значительно уменьшить качество пользовательского опыта;

3) *оптимизация контента*.

Правильная оптимизация контента для мобильных устройств считается основным условием эффективного взаимодействия с юзером. Это адаптация текстовых, а также зрительных элементов для экрана маленького размера, для того чтобы избежать горизонтальной прокрутки, наложений и слишком мелкого шрифта. Адаптивные алгоритмы также проводят проверку, в какой степени хорошо контент подстраивается под динамическое изменение экрана, к примеру, при повороте устройства. Веб-сайт обязан сохранять собственную работоспособность, а также удобочитаемость независимо от способа доступа, снабжая юзера простотой восприятия, а также взаимодействия.

Подобным способом автоматический анализ адаптивности веб-страниц дает возможность заранее обнаружить, а также устранить возможные трудности, связанные с применением мобильных устройств, а также существенно повысить общий пользовательский опыт. В условиях сегодняшнего рынка это считается никак не просто желательной, а нужной мерой с целью привлечения, а также удержания мобильных пользователей.

Применение и перспективы

Сайты для автоматической оценки качества веб-интерфейсов и распознавания их элементов находят широкое применение в различных областях:

1) веб-разработка: такие сайты помогают разработчикам быстро находить и устранять проблемы интерфейса на ранних стадиях разработки, что существенно сокращает время на тестирование и исправление ошибок;

2) UX-дизайн: для дизайнеров интерфейсов эти системы являются незаменимым инструментом, предоставляющим объективные данные для принятия решений по улучшению интерфейса;

3) цифровой маркетинг: в маркетинговых кампаниях улучшение пользовательского опыта на сайте может значительно повысить конверсию, удержание и лояльность клиентов;

4) образование: сайты для оценки интерфейсов могут использоваться в образовательных целях для обучения студентов основам UX-дизайна и анализа веб-ресурсов.

Заключение

Создание сайтов для автоматической оценки веб-интерфейсов и распознавания их элементов представляет собой важный шаг в направлении повышения качества цифровых продуктов. Эти технологии обладают значительным потенциалом и могут существенно улучшить процесс разработки и совершенствования интерфейсов.

Список литературы

1. Смирнов А. А., Петрова И. В. Анализ и улучшение UX веб-приложений. М. : ДМК Пресс, 2021. 320 с.

2. Кузнецова Л. В., Бондаренко Н. И. Оценка удобства использования мобильных интерфейсов // UX Design. 2023. № 6. С. 78–85.

3. Иванов И. И., Сидоров В. П. Машинное обучение для распознавания интерфейсов // Программная инженерия. 2022. № 4. С. 45−53.

4. Как идентифицировать компоненты интерфейса пользовательского интерфейса с помощью глубокого обучения? URL: https://underskyai.ru

Информация об авторах

Десятов Иван Вячеславович, студент, Пензенский государственный университет.

Свечников Артём Александрович, студент, Пензенский государственный университет.

Сорокин Алексей Александрович, доцент кафедры «Вычислительная техника», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004

ИСПОЛЬЗОВАНИЕ БЕЗДИСКОВОЙ СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ REDIS ДЛЯ УСКОРЕНИЯ ДОСТУПА КЛИЕНТОВ К МАССИВАМ ИЗМЕРИТЕЛЬНЫХ ДАННЫХ, ХРАНЯЩИХСЯ НА СЕРВЕРЕ

А. Л. Дмитриева[1], А. Р. Тарунин[2]

[1, 2]Мытищинский филиал Московского государственного технического университета им. Н. Э. Баумана, Мытищи, Россия

[1] aminaraon@gmail.com
[2] panterail@outlook.com

Аннотация. Представлено применение бездисковой системы управления базами данных Redis в качестве кеширующей базы данных Web-сервера, обслуживающего запросы нескольких клиентов к классической базе данных. Реализация позволяет существенно сократить время выполнения запросов при перекрытии интервалов в запрашиваемых данных в нескольких разных запросах. Результаты проведенных экспериментов показали, что наилучшей производительности удается достичь при использовании библиотеки Ioredis.

Ключевые слова: бездисковая СУБД Redis, кеширующая база данных Web-сервера, нагрузка на сервер, производительность системы, библиотека Ioredis

Для цитирования: Дмитриева А. Л., Тарунин А. Р. Использование бездисковой системы управления базами данных Redis для ускорения доступа клиентов к массивам измерительных данных, хранящихся на сервере // Вестник Пензенского государственного университета. 2024. № 4. С. 39–42.

Введение

Redis [1] – бездисковая система управления базами данных (СУБД), сохраняющая свои данные только в оперативной памяти, благодаря этому позволяет организовать очень быстрый поиск и выборку данных из своих баз данных (БД). В отличие от системы Memcached [2] Redis поддерживает организацию сохраняемых данных в несколько типов структур, включая хеши. Это свойство делает ее привлекательной для кеширования результатов выборки из БД классических СУБД (MySQL, MongoDB и др.).

В рамках научно-исследовательской работы студентов на кафедре К3 МФ МГТУ им. Н. Э. Баумана был создан Web-сервер webrobo [3], реализующий формы отображения зависящих от времени измерительных данных, накапливаемых в БД на отдельном сервере. В задачи сервера входит:

1) выдача по внешним Web-запросам клиентов форм отображения в формате html + javascript;

2) прием от загруженных форм запросов на данные из БД накопления в заданные интервалы времени;

3) синтез соответствующих запросов к серверу БД и выполнение этих запросов;

4) предварительная обработка данных, полученных на этапе (3), в частности, приведение к единой шкале измерений с применением градуировочных данных датчиков;

5) передача предобработанных данных запросившим их клиентам.

Выполнение пункта (3) может занимать значительное время. В случае многократного запроса от одного или нескольких клиентов данных из одного и того же диапазона времени желательно исключить повторные обращения к серверу БД. Эта задача может быть решена путем создания на сервере webrobo кеширующей БД под управлением СУБД Redis.

Метод решения

В качестве серверного программного обеспечения (ПО) webrobo используется система Node.js [4]. Эта система по умолчанию имеет готовую библиотеку доступа к базам данных СУБД Redis, которая так и называется: redis. При обработке запроса на получение данных за заданный интервал времени необходимо сначала выяснить, какие данные из этого интервала уже загружены в Redis, а уже затем синтезировать запросы к внешней БД для дополучения данных, которых в Redis еще нет.

Алгоритм проверки наличия данных в Redis для заданного временного интервала основан на сравнении запрошенного интервала с интервалами, уже хранящимися в Redis. Если интервал данных в Redis перекрывает запрошенный интервал, данные возвращаются напрямую из Redis. В противном случае алгоритм идентифицирует конкретные промежутки, отсутствующие в Redis, что позволяет запросить недостающие данные.

Пошаговый процесс выглядит следующим образом:

1. Получение запроса от клиента с интервалом времени.

2. Запрос к Redis и получение всех границ интервалов времени, данные которых сохранены в Redis.

3. Проверка пересечения запрошенного интервала с каждым из сохраненных в Redis, пока не закончатся сохраненные интервалы и/или они не перекроют полностью весь запрошенный интервал.

4. В случае если сохраненных интервалов недостаточно для покрытия запрошенного диапазона, вычисляются интервалы времени, отсутствующие в Redis, и запрашиваются данные за эти интервалы.

5. Полученные данные загружаются в Redis вместе с запрошенными интервалами. Это позволяет сократить избыточные запросы к внешнему серверу БД и ограничить их только недостающими интервалами.

6. Из Redis запрашиваются все данные, которые в дальнейшем проходят стадию предобработки и выдаются как ответ на запрос.

Таким образом, алгоритм проверки пересечений временных интервалов позволяет минимизировать нагрузку на сервер БД и обеспечить быстрый доступ к данным даже при массовых запросах.

Оптимизация взаимодействия с Redis

Проведенные испытания показали, что при большом объеме данных, циркулирующих в системе, использование Redis не дает ожидаемого эффекта ускорения повторного доступа к данным на сервере webrobo. Проведенные исследования показали, что одним из узких мест является применяемая библиотека redis. Одним из ключевых подходов к оптимизации стало применение специальной библиотеки ioredis [5], реализующей так называемый конвейер команд, что позволило повысить общую производительность сервера.

Конвейер команд (pipeline) – механизм, позволяющий отправить партию команд в Redis без ожидания ответа на каждую отдельную команду (рис. 1).

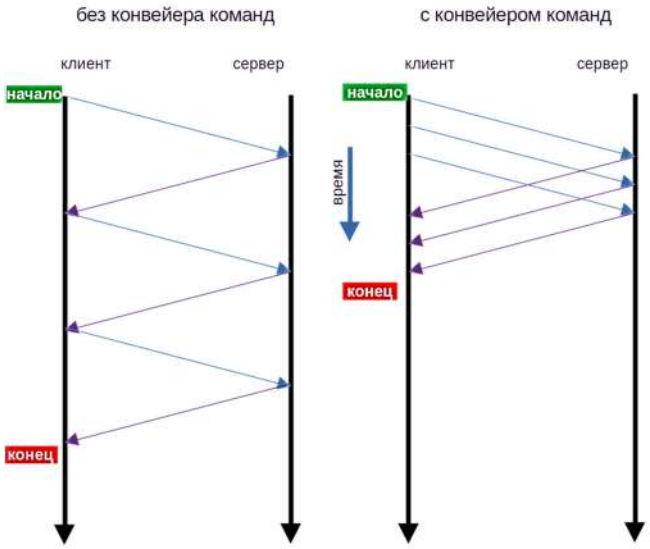
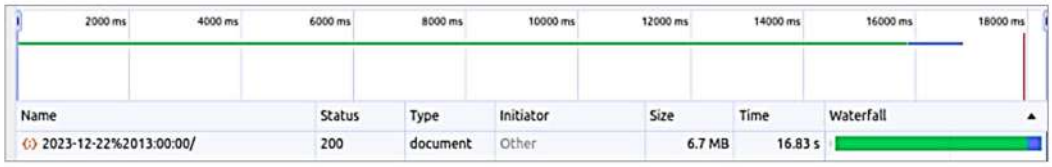


Рис. 1. Различие в передаче данных без использования и с использованием конвейеров

Конечный результат – повышенная устойчивость и оптимальная производительность системы.

Авторами были проведены тесты, результаты которых показаны на рис. 2.

Запрос № 1



Запрос № 2

Запрос № 3

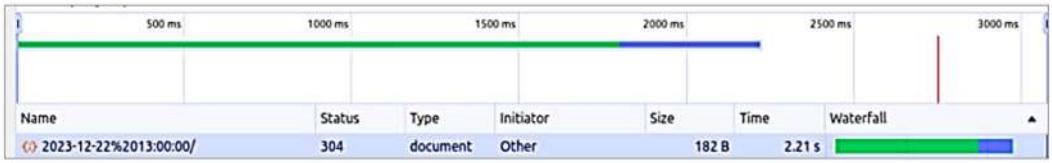


Рис. 2. Сравнение времени выполнения запросов

Запрос № 1 был выполнен с использованием библиотеки redis. Время выполнения запроса составило 16,83 с.

Запрос № 2 был выполнен с использованием оптимизированного кода и библиотеки ioredis, что дало заметный эффект: время выполнения запроса снизилось до 9,8 с. Стоит заметить, что запросы № 1 и № 2 выполнялись впервые после запуска сервера и, следовательно, происходило заполнение базы Redis данными, полученными из БД внешнего сервера.

Запрос № 3 был произведен сразу после запроса № 2, а значит, данные были взяты сразу из базы данных Redis, время выполнения составило 2,21 с. Статус ответа 304 указывает на то, что содержимое ответа не изменилось по сравнению с предыдущим запросом.

Заключение

В результате проведенной работы был реализован сервер, предоставляющий формы отображения измерительной информации webrobo, накапливаемой в БД внешнего сервера. За счет использования кеширующей БД на базе СУБД Redis и библиотеки ioredis взаимодействия системы Node.js с Redis удалось снизить нагрузку на сеть, обеспечивающую взаимодействие серверов, и в несколько раз уменьшить время выдачи данных клиентам, запрашивающим данные из пересекающихся интервалов времени.

Список литературы

1. Страница Redis Overview на сервере github.com. URL: https://github.com/redis (дата обращения: 31.10.2024).
2. Memcached. Официальный сайт. URL: https://memcached.org (дата обращения: 31.10.2024).
3. Главная страница сервера webrobo. URL: http://robo.itkm.ru/webrobo (дата обращения: 31.10.2024).
4. Node.js. Официальный сайт. URL: https://nodejs.org (дата обращения: 31.10.2024).
5. Страница ioredis на сайте npmjs.com. URL: https://www.npmjs.com (дата обращения: 31.10.2024).

Информация об авторах

Дмитриева Алина Леонидовна, студентка, Мытищинский филиал Московского государственного технического университета им. Н. Э. Баумана.

Тарунин Артём Романович, студент, Мытищинский филиал Московского государственного технического университета им. Н. Э. Баумана.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.43

ПРИМЕНЕНИЕ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ С ОГРАНИЧЕНИЯМИ ДЛЯ ЗАДАЧ ОПТИМИЗАЦИИ И ПЛАНИРОВАНИЯ

Л. Д. Зиновьев[1], Р. А. Каледа[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1] x.1.x2@yandex.ru
[2] kaledaroman@yandex.ru

Аннотация. Рассматривается применение логического программирования с ограничениями (CLP) на языке Prolog для решения задач оптимизации и планирования, востребованных в таких отраслях, как логистика, управление ресурсами и исследовательские проекты. Подчеркиваются преимущества Prolog в данной сфере (его декларативный подход, позволяющий описывать задачи на высоком уровне абстракции и фокусироваться на логике проблемы, минимизируя сложность алгоритмических деталей). Приводится классический пример задачи размещения N ферзей на шахматной доске, демонстрирующий выразительность и эффективность CLP в Prolog при решении комбинаторных задач. Обсуждаются ключевые принципы постановки ограничений и возможности расширения метода для адаптации к более сложным задачам оптимизации и планирования. Рассматриваются перспективы дальнейшего использования Prolog в реальных прикладных областях, где важно обеспечить гибкость и надежность решений при работе с большими объемами данных и сложными ограничениями.

Ключевые слова: оптимизация, ограничения, логистика, комбинаторика, ресурсы, декларативность, абстракция, алгоритмы, эффективность, гибкость, моделирование, решения

Для цитирования: Зиновьев Л. Д., Каледа Р. А. Применение логического программирования с ограничениями для задач оптимизации и планирования // Вестник Пензенского государственного университета. 2024. № 4. С. 43–45.

Современные задачи оптимизации и планирования становятся все более сложными, требуя инструментов, способных эффективно обрабатывать большие объемы данных и учитывать различные ограничения. Такие задачи особенно актуальны в логистике, управлении ресурсами и финансовом моделировании, где необходимы гибкие модели для сложных комбинаторных проблем. Традиционные подходы к программированию часто оказываются недостаточно гибкими, чтобы решать такие задачи с множеством условий.

Логическое программирование с ограничениями (Constraint Logic Programming, CLP) объединяет логический подход и управление условиями, предоставляя эффективные решения для разнообразных задач. Язык Prolog благодаря своей декларативной природе и встроенным средствам работы с ограничениями является одним из лучших инструментов для реализации CLP [1]. Он позволяет формулировать задачи на уровне логики, сосредотачиваясь на сути проблемы, что упрощает разработку и повышает читаемость кода.

Кроме того, Prolog обладает рядом преимуществ по сравнению с традиционными языками, такими как Python или Java. Декларативный подход в Prolog позволяет решать задачи на высоком уровне абстракции, встроенные механизмы ограничений обеспечивают быстрый поиск решений,

а гибкость языка позволяет легко адаптировать модели под изменяющиеся условия [2]. Эти качества делают Prolog востребованным инструментом для работы с задачами, где требуются высокая точность и адаптивность.

Логическое программирование с ограничениями в Prolog реализуется через библиотеку clpfd (Constraint Logic Programming over Finite Domains), которая позволяет работать с переменными в конечных доменах и накладывать на них различные ограничения [3]. Это обеспечивает эффективное решение комбинаторных задач, часто встречающихся в оптимизации и планировании.

Одной из классических задач, демонстрирующих возможности CLP в Prolog, является задача о N ферзях. Она заключается в размещении N ферзей на шахматной доске размером N × N таким образом, чтобы ни один ферзь не атаковал другого. Эта задача хорошо иллюстрирует принципы постановки ограничений и поиска решений в Prolog.

Реализация задачи N ферзей в Prolog с использованием CLP представлена на рис. 1.

```
:- use_module(library(clpfd)).

n_queens(N, Solution) :-
    length(Solution, N),
    Solution ins 1..N,
    safe_queens(Solution),
    label(Solution).

safe_queens([]).
safe_queens([Q|Qs]) :-
    safe_queens(Qs),
    no_attack(Q, Qs, 1).

no_attack(_, [], _).
no_attack(Q, [Q1|Qs], D) :-
    Q #\= Q1,
    Q #\= Q1 + D,
    Q #\= Q1 - D,
    D1 #= D + 1,
    no_attack(Q, Qs, D1).
```

Рис. 1. Реализация задачи N ферзей на языке программирования Prolog

Данный код использует библиотеку clpfd для задания ограничений на переменные. Переменная Solution представляет собой список длины N, где каждый элемент соответствует позиции ферзя в строке. Ограничения обеспечивают, что ферзи не находятся в одном столбце и не атакуют друг друга по диагоналям. Предикат label (Solution) запускает процесс поиска решений, выполняя перебор значений переменных в Solution в соответствии с заданными ограничениями. Он просматривает возможные варианты и присваивает конкретные значения каждой переменной, что позволяет найти все корректные комбинации расстановки ферзей на шахматной доске.

Далее предикат safe_queens (Solution) отвечает за проверку безопасности расположения ферзей, чтобы они не угрожали друг другу по диагонали. Он рекурсивно разбирает список Solution, где каждый элемент представляет ферзя в своей строке. Вызов safe_queens (Qs) последовательно проверяет остальные ферзи в списке, обеспечивая их расположение таким образом, чтобы они не могли атаковать друг друга.

Ключевую роль здесь играет предикат no_attack (Q, Qs, D), который выполняет проверку для текущего ферзя Q относительно оставшихся ферзей в списке Qs. Параметр D – это расстояние

(в строках) между Q и каждым из оставшихся ферзей. Условия проверяют, что ферзь Q не находится на одной диагонали с каждым ферзем Q1 из списка Qs. Увеличение переменной D на единицу при каждом вызове предиката no_attack гарантирует смещение вниз по строкам.

Prolog – это язык программирования, который отлично подходит для решения задач с множеством ограничений. Главное преимущество этого языка – простой способ описать, что нужно сделать, а не как именно [4]. Вместо того, чтобы писать сложные алгоритмы, разработчики просто указывают правила и связи между элементами.

В других высокоуровневых языках для работы с концепцией Constraint Logic Programming (CLP) приходится писать большое количество кода. В языке Prolog уже есть встроенные инструменты для этого. Также в большинстве императивных языков программисту самому нужно следить за путем выполнения кода, что усложняет работу и делает разработку менее гибкой. В Prolog многое автоматизировано, поэтому можно сразу фокусироваться на решении задачи.

Таким образом, программирование с ограничениями дает много преимуществ в задачах оптимизации и планирования. Высокий уровень абстракции позволяет описывать задачи проще и понятнее. Например, задача с расстановкой ферзей на шахматной доске показывает, как Prolog решает сложные задачи с множеством вариантов.

Кроме того, Prolog легко адаптируется под новые требования. Это важно для реальных проектов, в которых условия могут стремительно меняться. Prolog помогает оптимизировать маршруты и управлять ресурсами, а в сфере финансов – строить точные прогнозы. Возможность быстро вносить изменения делает этот язык ценным инструментом для разработчиков.

Prolog является отличным вариантом для программирования с ограничениями. Эти особенности делают его не только интересным с теоретической, но и ценным с практической точки зрения для специалистов в области функционального программирования.

Список литературы

1. Constraint Logic Programming. URL: https://www.swi-prolog.org (дата обращения: 20.10.2024).

2. Росси Ф., Ван Бик П., Уолш Т. Справочник по программированию с ограничениями. Эльзевир, 2006. С. 115–130. URL: https://www.sciencedirect.com (дата обращения: 21.10.2024).

3. Jaffar J., Maher M. J. Constraint Logic Programming: A Survey // Journal of Logic Programming. 1994. Vol. 19, № 20. URL: https://www.sciencedirect.com (дата обращения: 24.10.2024).

4. Шрайбер П. А. Основы программирования на языке Пролог: курс лекций. М. : Просвещение/Бином, 2017. С. 127–146.

Информация об авторах

Зиновьев Леонид Дмитриевич, студент, Пензенский государственный университет.

Каледа Роман Александрович, студент, Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 378.004

ВЕРИФИКАЦИЯ ИГРОВЫХ УРОВНЕЙ UNITY С ПОМОЩЬЮ АЛГОРИТМА ЛИ

Е. С. Ильинская [1], В. В. Эпп [2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1] ilinskaya58k@mail.ru
[2] vitalinae@mail.ru

Аннотация. В целях разработки программного комплекса генерации и верификации уровней на мультиплатформенном движке Unity и дальнейшего их применения были проанализированы и протестированы встроенные возможности движка Unity, интеграции языка программирования C# в среду разработки проекта, а также были проведены исследования основного алгоритма верификации, который будет задействован в разработке. Основной упор делался на применение алгоритма Ли как основного средства автоматической верификации и тестирования. В качестве основного инструмента генерации уровней были задействованы базовые методы и возможность движка Unity.

Ключевые слова: верификация, Unity, генерация, 2D, C#, алгоритм Ли, компьютерные игры

Для цитирования: Ильинская Е. С., Эпп В. В. Верификация игровых уровней UNITY с помощью алгоритма Ли // Вестник Пензенского государственного университета. 2024. № 4. С. 46–49.

Введение

В настоящее время игры занимают важное место в жизни многих людей, и разработчики игр постоянно стараются улучшить качество их продуктов. Одним из ключевых аспектов игрового процесса является создание уровней, которые должны быть интересными и вызывающими у игроков желание продолжать играть. Однако создание уровней – это сложный процесс, который требует много времени и труда. В этом контексте возникает необходимость верификации уровней, чтобы убедиться, что они соответствуют заданным требованиям и не содержат ошибок. В данной работе рассматривается использование алгоритма Ли для верификации игровых уровней, что позволяет автоматизировать процесс и повысить качество создаваемых уровней.

Алгоритм Ли позволяет автоматически проверять уровни на наличие пути от начала до конца, а также на наличие недостижимых областей и других ошибок в дизайне уровня. Это значительно упрощает процесс верификации и повышает качество игрового контента. Кроме того, использование алгоритма может помочь ускорить процесс создания уровней, так как автоматическая проверка может выявлять ошибки и предупреждать разработчиков заранее. Также стоит отметить, что использование алгоритма Ли для верификации игровых уровней может существенно сократить время, затрачиваемое на создание уровней.

Задачей разработки является создание программы для случайной генерации игровых уровней, которые будут проверяться на проходимость, с помощью алгоритма Ли.

Алгоритм Ли – это алгоритм поиска пути в лабиринте, который использует волновую функцию. Он был разработан Ли Харви в 1961 г. и с тех пор нашел широкое применение в робототехнике, компьютерных играх и других областях [1].

Алгоритм может быть использован для поиска пути в лабиринтах любой сложности и формы и не требует больших вычислительных мощностей. Кроме того, алгоритм Ли может быть легко модифицирован для учета препятствий и других условий.

Создание игр на Unity является удобным и популярным выбором для многих разработчиков. Во-первых, Unity предоставляет широкий спектр инструментов и ресурсов для создания игр, включая графический движок, физический движок, аудиосистему, инструменты моделирования и анимации, а также множество готовых компонентов и библиотек.

Во-вторых, Unity обладает простым и интуитивно понятным интерфейсом, что делает процесс создания игры более доступным для начинающих разработчиков.

В-третьих, Unity поддерживает множество платформ, включая PC, мобильные устройства, консоли и виртуальную реальность, что позволяет создавать игры для широкой аудитории.

Кроме того, Unity имеет большое сообщество разработчиков, которые делятся своими знаниями и опытом, а также предоставляют готовые решения и инструменты для ускорения процесса разработки [2, 3].

Концепция алгоритма Ли

Алгоритм Ли, также известный как алгоритм распространения волны, используется для поиска кратчайшего пути в лабиринте. Он начинает работу с заданной начальной точки и распространяет волну от нее до тех пор, пока не достигнет конечной точки. В процессе распространения волны каждая клетка лабиринта помечается расстоянием от начальной точки, что позволяет найти кратчайший путь.

Алгоритм Ли работает следующим образом:

1. Задаются начальная и конечная точки в лабиринте.

2. Начальная точка помечается единицей, а все остальные клетки лабиринта помечаются нулем, или NULL.

3. На первой итерации алгоритма, клетки, соседствующие с начальной точкой, помечаются двойкой.

4. На следующей итерации клетки, соседствующие с уже помеченными единицей клетками, помечаются тройкой.

5. Процесс продолжается до тех пор, пока волна не достигнет конечной точки. Каждая клетка помечается расстоянием от начальной точки.

6. Когда волна достигнет конечной точки, алгоритм останавливается и кратчайший путь находится обратным ходом от конечной точки к начальной, следуя по клеткам с наименьшим расстоянием.

На выходе получаем значение, равное длине пути, а также маршрут от источника к приемнику.

Для устранения неопределенности проведения трассы в случае, если несколько соседних площадок имеют одинаковый вес, используются путевые координаты. Эти координаты указывают предпочтительное направление трассы.

Пример работы алгоритма Ли (рис. 1).

На данном рисунке представлено поле, которое подается на вход. Белым обозначены проходимые (свободные) клетки, зеленым – непроходимые (занятые). Точка А является источником, т.е. начальной точкой, от которой будет распространяться волна. Точка Б – приемник, т.е. конечная точка, к которой следует проложить маршрут.

Когда волна достигнет конечной точки, алгоритм останавливается и кратчайший путь находится обратным ходом от конечной точки к начальной, следуя по клеткам с наименьшим расстоянием.

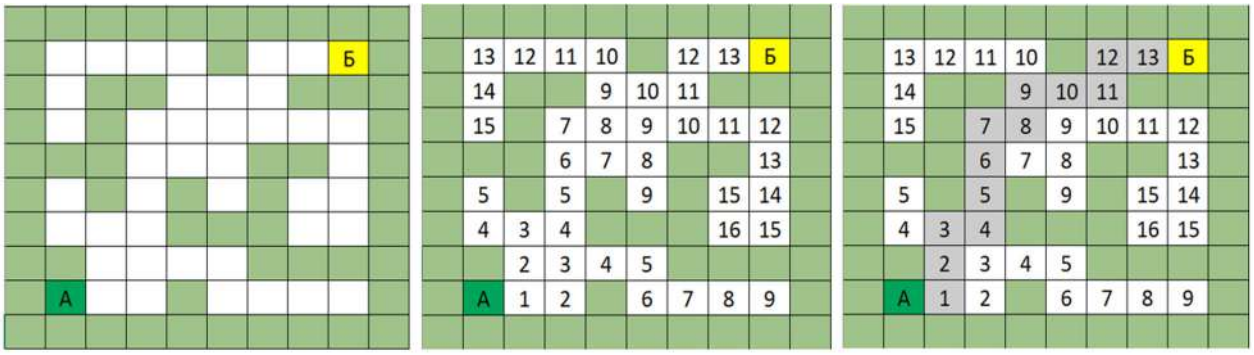


Рис. 1. Лабиринт

Разработка

Для генерации уровня пользователю предоставлена возможность настройки параметров, таких как размеры поля, количество лавовых озер, луж и ям и т.д.

При запуске приложения для этих параметров установлены значения по умолчанию, которые можно изменять по мере необходимости. Для всех параметров предусмотрена проверка на допустимые значения.

Для визуализации алгоритма Ли также предусмотрена возможность запуска поиска безопасного пути. В случае отсутствия безопасного маршрута доступен альтернативный путь, включающий прохождение через клетки типа «HotRock» (горячий пол). Если безопасный путь не найден и альтернативный маршрут также неудовлетворителен, рекомендуется перегенерировать уровень, чтобы избежать нежелательных блоков.

На рис. 2 изображена визуализация маршрута от точки А к точке Б на сгенерированном игровом уровне.

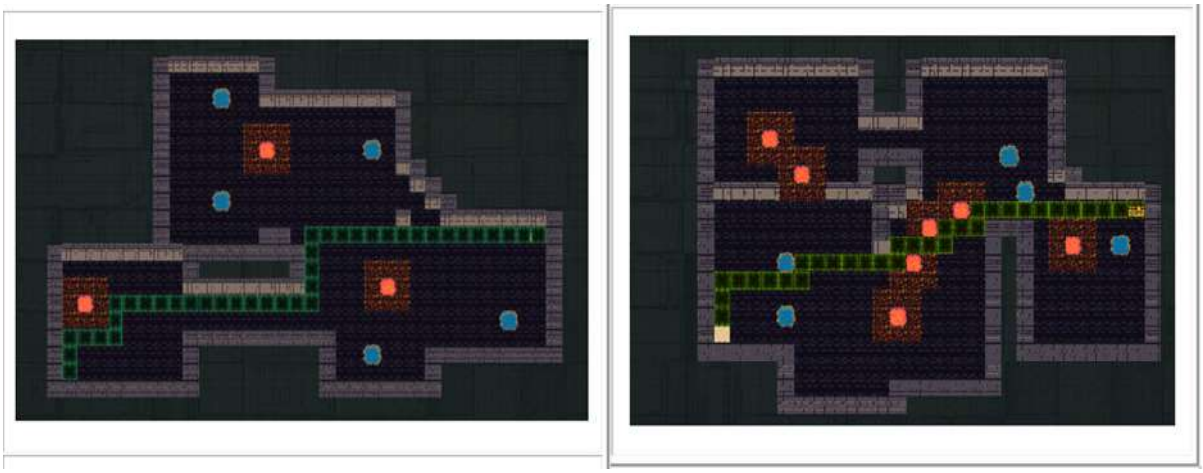


Рис. 2. Построение пути на игровом уровне

Если путь не может быть построен безопасным способом, то у пользователя, все еще есть возможность проложить альтернативный маршрут.

Под альтернативным маршрутом понимается тот маршрут, который подразумевает прохождение по клеткам «HotRok» (горячий пол). Однако, если прохождение уровня невозможно безопасным маршрутом, то рекомендуется перегенерировать уровень таким образом, чтобы не затрагивать желательные для избегания блоки.

Заключение

Таким образом, используя волновой алгоритм поиска пути, можно получить инструмент для автоматической верификации игровых уровней, что позволит сократить время, затрачиваемое на создание и тестирование уровней [4, 5].

Список литературы

1. Информационная система Wikipedia : рабочая программа. URL: https://ru.wikipedia.org (дата обращения: 05.05.2023).

2. Ларкович С. Н., Евдокимов П. В. C# для UNITY-разработчиков. Практическое руководство по созданию игр. СПб. : Наука и техника, 2023. 368 с.

3. Бонд Дж. Г. Unity и C#. Геймдев от идеи до реализации. 2-е изд. СПб. : Питер, 2019. 928 с.

4. Гниденко И. Г., Павлов Ф. Ф., Фёдоров Д. Ю. Технологии и методы программирования. М. : Юрайт, 2022. 236 с.

5. Мартин Р. Чистый код: создание, анализ и рефакторинг. СПб. : Питер, 2019. 464 с.

Информация об авторах

Ильинская Екатерина Сергеевна, магистрант, Пензенский государственный университет.

Эпп Виталина Викторовна, кандидат технических наук, доцент, доцент кафедры «Системы автоматизированного проектирования», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 378.004

ПРИМЕНЕНИЕ ПРИНЦИПОВ SOLID ПРИ СОЗДАНИИ ИГР НА UNITY

Е. С. Ильинская[1], В. В. Эпп[2], С. С. Балясников[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1] ilinskaya58k@mail.ru
[2] vitalinae@mail.ru
[3] lolziz12we@gmail.com

Аннотация. В целях изучения были рассмотрены принципы SOLID, их специфика применения при разработке на мильтиплатформенном движке UNITY. В процессе изучения и анализа были сформулированы основные сложности при работе с принципами SOLID, а также типичные ошибки, с которыми нередко сталкиваются начинающие разработчики. Для применения принципов SOLID в разработке на движке UNITY были приведены примеры реализации под конкретное задание или ситуацию, а также были указаны примеры нарушения принципов SOLID.

Ключевые слова: SOLID, разработка, UNITY

Для цитирования: Ильинская Е. С., Эпп В. В., Балясников С. С. Применение принципов SOLID при создании игр на UNITY // Вестник Пензенского государственного университета. 2024. № 4. С. 50–53.

Введение

В 2024 г. в Нижнем Новгороде состоялся итоговый этап по компетенции «Разработка компьютерных игр и мультимедийных приложений» в рамках Чемпионата по профессиональному мастерству «Профессионалы–2024», на котором собрались эксперты-наставники и конкурсанты с разных уголков нашей страны, чтобы продемонстрировать свои навыки программирования в самых различных областях. Одной из самых сложных и одновременно интересных компетенций стала «Разработка компьютерных игр».

Чемпионат «Профессионалы–2024» – это уникальная возможность для студентов среднего профессионального образования проявить свои таланты, продемонстрировать готовность решать сложные задачи и узнать много нового в своей профессиональной сфере.

Каждая компетенция была продумана до мелочей, таковой стала и «Разработкой компьютерных игр и мультимедийных приложений». Заданием для участников было создание полноценной игры в жанре «стратегия». Этот жанр предъявляет особые требования к планированию игрового процесса, логике взаимодействий и глубокой проработке.

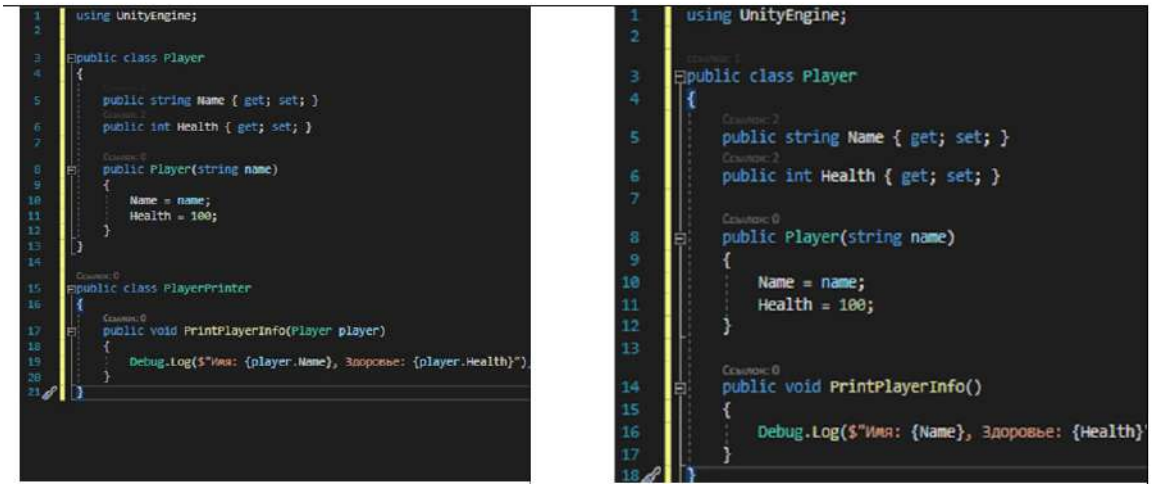
В результате у конкурсантов должна была получиться игра в жанре «стратегия», которая должна была соответствовать требованиям стандарта программирования, а также учитывать принципы SOLID.

SOLID – это набор из пяти принципов объектно-ориентированного программирования. Применение этих подходов помогает создавать программный код, который можно легко модифицировать и дополнять.

Рассмотрим некоторые принципы, которые необходимо учитывать при разработке игр и которые были важны на конкурсном задании.

Принцип единственной ответственности (SRP)

Каждый класс должен иметь только одну причину для изменения (рис. 1).



```
using UnityEngine;

public class Player
{
    public string Name { get; set; }
    public int Health { get; set; }

    public Player(string name)
    {
        Name = name;
        Health = 100;
    }
}

public class PlayerPrinter
{
    public void PrintPlayerInfo(Player player)
    {
        Debug.Log($"Имя: {player.Name}, Здоровье: {player.Health}");
    }
}
```

```
using UnityEngine;

public class Player
{
    public string Name { get; set; }
    public int Health { get; set; }

    public Player(string name)
    {
        Name = name;
        Health = 100;
    }

    public void PrintPlayerInfo()
    {
        Debug.Log($"Имя: {Name}, Здоровье: {Health}");
    }
}
```

а) *б*)

Рис. 1. Принцип единственной ответственности (SRP):
а – использование принципа, *б* – нарушение принципа

В примере класс «Player» отвечает только за управление данными игрока, такими как имя и здоровье. Класс «PlayerPrinter» отдельно отвечает за вывод информации об игроке. Это разделение позволяет легко модифицировать одну часть, не затрагивая другую, что и является сутью принципа единственной ответственности.

Конкурсанты сталкивались с трудностями в определении, что именно является «единственной» ответственностью класса. Это требует опыта и понимания, как разделять функциональность, чтобы избежать создания классов с множественными обязанностями.

Принцип открытости/закрытости (OCP)

Классы должны быть открыты для расширения, но закрыты для модификации (рис. 2).

```
using UnityEngine;

public abstract class Weapon
{
    public abstract void Attack();
}

public class Sword : Weapon
{
    public override void Attack()
    {
        Debug.Log("Меч атакует!");
    }
}

public class Bow : Weapon
{
    public override void Attack()
    {
        Debug.Log("Лук стреляет!");
    }
}
```

```
using UnityEngine;

public class Weapon
{
    public void Attack(string weaponTyp
    {
        if (weaponType == "Sword")
        {
            Debug.Log("Меч атакует!");
        }
        else if (weaponType == "Bow")
        {
            Debug.Log("Лук стреляет!");
        }
    }
}
```

а) *б*)

Рис. 2. Принцип открытости/закрытости (OCP):
а – использование принципа, *б* – нарушение принципа

Абстрактный класс Weapon позволяет добавлять новые виды оружия, такие как Sword и Bow, без изменения существующего кода.

Принцип подстановки Лисков (LSP)

Объекты суперкласса должны быть заменяемы объектами подкласса (рис. 3).

а) *б*)

Рис. 3. Принцип подстановки Лисков (LSP):
а – использование принципа, *б* – нарушение принципа

В данном коде Goblin может быть использован везде, где ожидается Enemy, без нарушения логики программы.

В отношении соблюдения предусловий и постусловий у конкурсантов также были проблемы с соблюдением этого принципа.

Заключение

Применение решения SOLID при создании игры в жанре стратегии оказалось ключевой стороной, которую не всем конкурсантам удалось успешно решить. Однако участие в таких конкурсах позволяет развивать не только технические навыки, но и умение структурно мыслить и решать сложные задачи в короткие сроки [1–4].

Список литературы

1. Ларкович С. Н., Евдокимов П. В. C# для UNITY-разработчиков. Практическое руководство по созданию игр. СПб. : Наука и Техника, 2023. 368 с.

2. Гниденко И. Г., Павлов Ф. Ф., Фёдоров Д. Ю. Технологии и методы программирования. М. : Юрайт, 2022. 236 с.

3. Мартин Р. Чистый код: создание, анализ и рефакторинг. СПб. : Питер, 2019. 464 с.

4. Бонд Дж. Г. Unity и C#. Геймдев от идеи до реализации. 2-е изд. СПб. : Питер, 2019. 928 с.

Информация об авторах

Ильинская Екатерина Сергеевна, магистрант, Пензенский государственный университет.

Эпп Виталина Викторовна, кандидат технических наук, доцент, доцент кафедры «Системы автоматизированного проектирования», Пензенский государственный университет.

Балясников Станислав Сергеевич, студент, Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.62

ОЗЁРА ДАННЫХ

И. А. Казакова

Пензенский государственный университет, Пенза, Россия

kia-2011@yandex.ru

Аннотация. Рассмотрено применение озёр данных – хранилищ неструктурированных данных, получаемых из различных источников. Рассмотрены возможность совместного использования хранилищ и озёр данных, которая позволяет повысить эффективность использования «сырых», необработанных данных; программное обеспечение для работы с озёрами данных.

Ключевые слова: данные, озеро данных, неструктурированные данные, хранилище данных, масштабируемость, программное обеспечение

Для цитирования: Казакова И. А. Озёра данных // Вестник Пензенского государственного университета. 2024. № 4. С. 54–56.

Аналитики крупных предприятий и компаний часто работают с неструктурированными и разнородными данными. Это могут быть данные сайтов, данные опросов клиентов, POS-систем, CRM-систем, записи камер в залах ожидания, социальные сети, интернет-магазины, банковские приложения, информация, полученная с датчиков, т.е. данные любых бизнес-систем. Все эти данные нужно хранить, чтобы впоследствии использовать для анализа и выработки решения. Для того, чтобы загрузить такие «сырые» данные в базу данных, требуются значительные затраты.

Легче решается эта проблема с помощью data lake – озёр данных. Они дают возможность быстро и эффективно работать с огромными объемами неструктурированных данных в исходном виде. Одно из самых известных определений озера данных от специалиста по разработке программного обеспечения М. Фаулера: «Озеро данных – это файловое хранилище всех типов сырых данных, которые доступны для анализа кем угодно в организации» [1].

Озеро данных принимает данные любых форматов. Источник данных не имеет значения. Принимаются и хранятся все данные, а структура данных учитывается только в момент извлечения для определенной задачи. Поступающей в озеро информации присваиваются метаданные: время поступления, источник, формат, структура и др. Особенностями озёр являются продолжительный срок данных хранения необработанных данных, возможность реорганизации данных и использования различных схем доступа к данным.

Заранее невозможно определить объем данных, которые будут храниться в озере данных. Поэтому эта архитектура озера данных обеспечивает расширенную масштабируемость до эксабайта (2^{60} байт). Обычная система хранения такого обеспечить не может.

Эффективно совместное использование озёр данных с хранилищами данных. Если хранящиеся в озере данные требуются для ответа на конкретный бизнес-вопрос, их можно извлечь, преобразовать и загрузить в хранилище данных для последующего анализа [1].

Озёра данных так же, как и хранилища данных, создаются с одинаковой целью – хранение данных для поддержки анализа данных и принятия решений. Хранилища и озёра данных отличаются своей архитектурой. Сравнительная характеристика архитектур приведена в табл. 1.

Таблица 1

Сравнительная характеристика архитектур озёр и хранилищ данных

	Озеро данных	Хранилище данных
Структура	Схема не задается заранее. Данные обрабатываются во время использования. Это схема при чтении –Schema On Read (т.е. при анализе данных) [2]	Схема задается заранее, до записи данных. Это Schema On Write – схема при записи
Данные	Любой тип данных из любого источника	Структурированные
Гибкость	Гибкие и адаптируемые к изменениям в использовании	Схему нельзя быстро изменить в соответствии с меняющимися требованиями
Доступ к данным	Требуются определенные навыки из-за неопределенности схемы	Легко доступны благодаря структурированной, определенной схеме
Стоимость хранения	Низкая стоимость	Высокая стоимость
Основные пользователи	Специалисты по глубокому анализу данных	Оперативные пользователи
Получение данных	Высокая скорость	Низкая скорость

Для работы с озёрами данных следует наладить процесс управления данными. До загрузки в озеро необходимо проверить качество и надежность данных. Этого можно достичь различными способами, например:

– не принимать данные из сомнительных источников;

– исключить доступ к загрузке данных сотрудникам, у которых нет на это прав;

– ограничить возможность загрузки в озеро графических файлов большого объема.

Для озёр данных разработано программное обеспечение, которое упорядочивает данные в озере и упрощает доступ к данным и их использование. Самое известное – Hadoop –технология с открытым исходным кодом от компании Apache [3]. Экосистема Hadoop состоит из трех основных элементов:

1) распределенная файловая система HDFS, основная функция которой – хранение и репликация данных на нескольких серверах;

2) инструмент управления ресурсами (YARN);

3) инструмент разделения данных на более мелкие части перед обработкой на серверах.

Дополнительные инструменты облегчают процессы сбора, подготовки и извлечения данных. Hadoop доступен для работы с облачными корпоративными платформами для создания облачного озера данных. Он обеспечивает более быстрые вычисления, что сделало его достаточно популярным.

Озёра широко применяются для хранения интересной информации, которая временно не используется для аналитики, или для данных, которые в настоящий момент не нужны, но, возможно, будут нужны в будущем.

Таким образом, озёра данных находят применение для глубокого анализа данных и формирования гипотез. Они позволяют собрать большое количество данных, чтобы затем с помощью инструментов машинного обучения, аналитики и искусственного интеллекта сравнивать различные факты, строить всевозможные прогнозы, анализировать информацию со всех сторон и извлекать из данных максимум пользы.

Список литературы

1. Фаулер М. Озеро данных. URL: https://martinfowler.com (дата обращения: 31.10.2024).

2. Шпрингер Е. Что такое озёра данных и почему в них дешевле хранить big data. URL: https://cloud.vk.com (дата обращения: 31.10.2024).

3. Что такое схема при чтении и схема при записи в Hadoop? URL: https://www.geeksforgeeks.org (дата обращения: 31.10.2024).

Информация об авторе

Казакова Ирина Анатольевна, доцент, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Автор заявляет об отсутствии конфликта интересов.

УДК 004.42

СРАВНЕНИЕ РЕКУРСИВНОЙ И ИТЕРАТИВНОЙ РЕАЛИЗАЦИИ АЛГОРИТМА ПОИСКА В ГЛУБИНУ

А. А. Калугин[1], Л. В. Гурьянов[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1]kartem423@gmail.com
[2]leo8087@yandex.ru

Аннотация. Алгоритм поиска в глубину, или DFS (Depth-first search), – это один из основных и часто используемых графовых алгоритмов. Рассматриваются рекурсивная и итеративная реализации этого алгоритма. Обосновывается выбор языка программирования алгоритма. Приводятся статистические результаты тестирования алгоритма на графах с количеством вершин до 2000.

Ключевые слова: алгоритм поиска в глубину, рекурсивная реализация, итеративная реализация

Для цитирования: Калугин А. А., Гурьянов Л. В. Сравнение рекурсивной и итеративной реализации алгоритма поиска в глубину // Вестник Пензенского государственного университета. 2024. № 4. С. 57–59.

Введение

Алгоритм поиска в глубину, или DFS (Depth-first search), – это один из основных и часто используемых графовых алгоритмов [1]. Алгоритм заключается в обходе графа внутрь, начиная с некоторой вершины до тех пор, пока у текущей вершины не будут посещены все соседи. Далее возвращаемся назад до тех пор, пока у начальной вершины не останется непосещенных вершин.

Целей использования данного алгоритма существует великое множество, начиная от обычного подсчета глубины вершины и заканчивая топологической сортировкой.

Рекурсивная реализация поиска в глубину

Наиболее распространенной реализацией поиска в глубину является рекурсивная. Ее суть заключается в вызове для начальной вершины s функции (назовем ее dfs(s)), в которую передается номер этой вершины. При этом сама функция реализует следующий алгоритм:

1) пометить текущую вершину посещенной;
2) (опционально) совершить некоторые дополнительные действия;
3) для всех непосещенных соседних вершин вызвать функцию dfs;
4) (опционально) совершить некоторые дополнительные действия;
5) вернуть результат.

Из плюсов такой реализации можно выделить простоту кодирования функции dfs, простоту получения возвращаемых данных и поддержки порядка обхода графа.

Итеративная реализация поиска в глубину

Итеративная реализация является более экзотичной ввиду наличия более простой и распространенной рекурсивной реализации. Однако несомненным плюсом итеративной реализации является отсутствие необходимости делать длинный стек вызовов функции, который необходим для рекурсивного подхода.

Для некоторых языков программирования этот плюс может оказаться решающим. Так, например, Python очень плохо обрабатывает рекурсию, из-за чего в этом языка даже существует ограничение на ее глубину. В случае, если снять это ограничение, время и память, необходимые для глубокой рекурсии, настолько огромны, что уже при 10 000 вызовов программа зависнет. В свою очередь, при итеративном подходе Python может обработать глубину в 100 000 вызовов.

Минусом итеративной реализации является необходимость вручную поддерживать стек обхода графа, что, в свою очередь, усложняет реализацию.

Для более точного сравнения двух подходов реализации DFS было решено выбрать язык, более подходящий как для рекурсии, так и для итерации. В качестве такого языка программирования был выбран язык C++ 20.

Сравнительный анализ реализаций DFS выполнен для графов с количеством вершин от 5 до 2000. Результаты сравнения представлены в табл. 1 и на рис. 1, 2.

Таблица 1

Результаты тестирования

N (количество вершин)	Метод	
	Итеративная реализация (время/память)	Рекурсивная реализация (время/память)
5	2 мс, 3 Кб	4 мс, 5 Кб
15	3 мс, 5 Кб	5 мс, 8 Кб
100	20 мс, 15 Кб	18 мс, 20 Кб
1000	81 мс, 60 Кб	92 мс, 61 Кб
2000	124 мс, 60 Кб	154 мс, 68 Кб

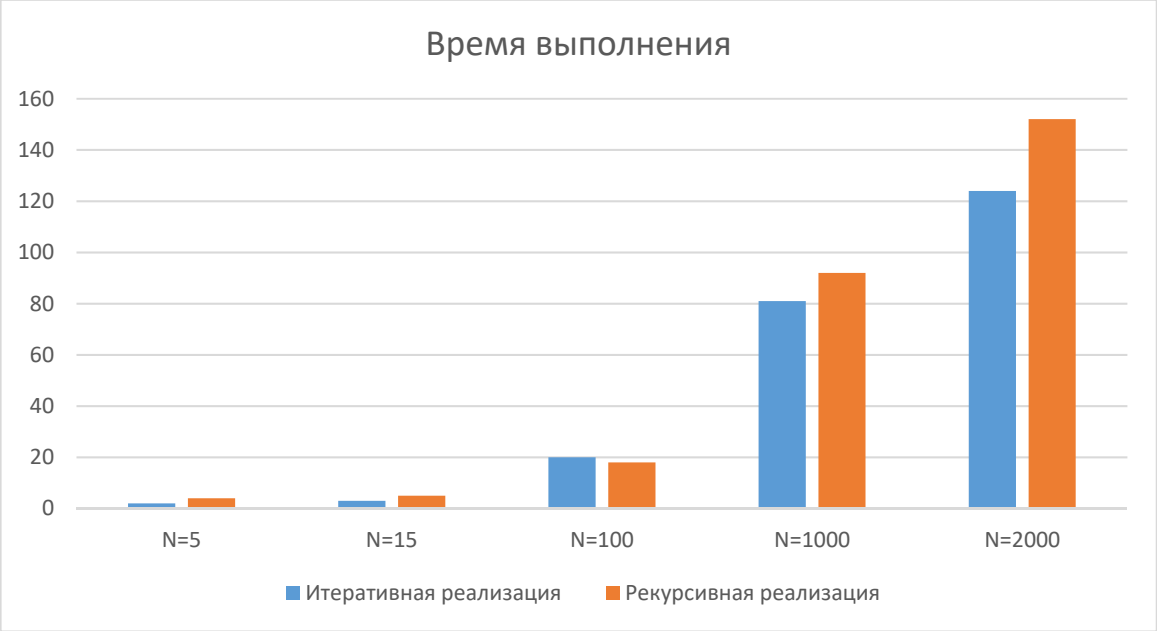


Рис. 1. Результаты тестирования (время выполнения в мс)

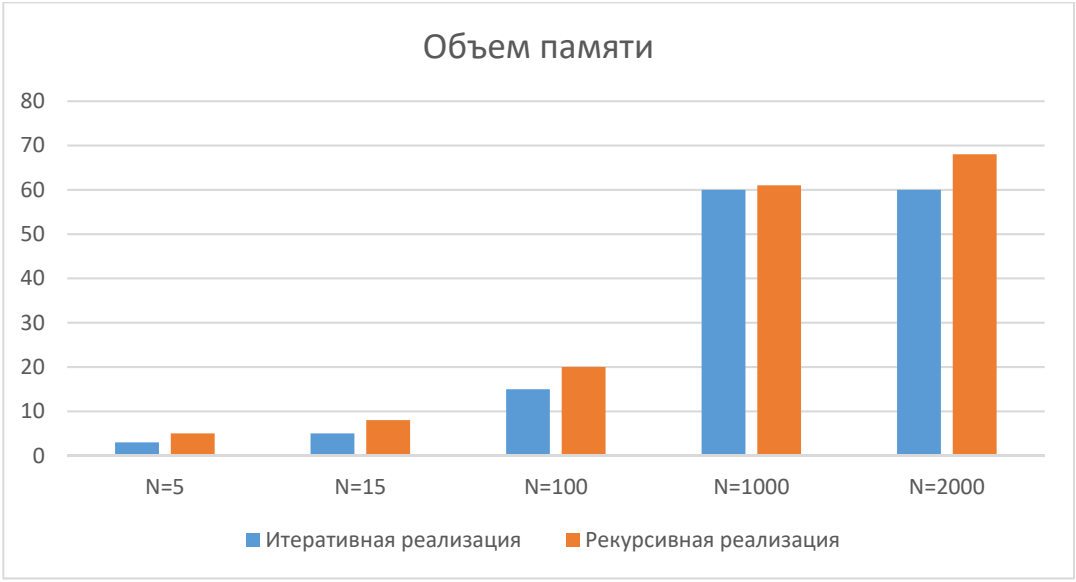


Рис. 2. Результаты тестирования (объем памяти в Кб)

Как видно из данных таблицы и рисунков, на данных тестах в среднем итеративная реализация DFS показывает лучшие результаты по времени выполнения и объему памяти.

Заключение

Можно сделать вывод, что в общем случае можно использовать рекурсивный подход из-за его простоты реализации и неплохой эффективности. Если же ускорение на 20–30 % может оказаться решающим, имеет смысл использовать итеративный подход.

Список литературы

1. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ : пер. с англ. 2-е изд. М. : Вильямс, 2011. 1296 с.

Информация об авторах

Калугин Артём Андреевич, студент, Пензенский государственный университет.

Гурьянов Лев Вячеславович, кандидат технических наук, доцент, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.432

МАКРОСЫ В ЯЗЫКЕ LISP

А. А. Кареев[1], А. Г. Михалев[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1] andrey.andreych@bk.ru
[2] mag@pnzgu.ru

Аннотация. Макросы в языке программирования LISP представляют собой мощный инструмент для создания синтаксических абстракций и метапрограммирования. Возможности, предоставляемые макросами, значительно упрощают написание и поддержку сложных программ.

Ключевые слова: LISP, логическое программирование, макросы, метапрограммирование

Для цитирования: Кареев А. А., Михалев А. Г. Макросы в языке LISP // Вестник Пензенского государственного университета. 2024. № 4. С. 60–62.

Макросы – это функции, которые получают исходные выражения в виде кода и возвращают новый код. Они выполняются на этапе компиляции, что позволяет контролировать и изменять структуру программы еще до ее выполнения. В отличие от функций, которые работают с вычисленными значениями, макросы работают с самими выражениями, что позволяет им изменять синтаксические конструкции.

Макросы позволяют создавать новые элементы синтаксиса, которые делают код более читаемым и понятным. Например, в LISP отсутствуют такие конструкции, как while или unless, но с помощью макросов их легко добавить. Благодаря этому разработчики могут создавать абстракции, специфичные для задачи, делая код более выразительным и интуитивно понятным. В языках, таких как Java или Python, расширение синтаксиса требует сложного обходного пути, часто с использованием фабрик или классов. Макросы LISP позволяют добавлять конструкции к языку непосредственно, что упрощает разработку и улучшает читаемость кода. Макросы в LISP предоставляют возможности метапрограммирования, позволяя фрагментам кода быть самодостаточными и расширяемыми. Это значит, что программа может создавать свой собственный код в зависимости от входных данных. Например, макросы могут автоматически генерировать повторяющиеся или сложные конструкции, избавляя разработчика от рутины.

Благодаря тому, что макросы выполняются на этапе компиляции, они помогают избежать накладных расходов во время выполнения программы [1]. Например, сложные проверки условий или часто используемые выражения могут быть оптимизированы с помощью макросов, так как компилятор их выполняет один раз. Это особенно полезно в системах, требующих высокой производительности. Далее показано, как можно добавить собственную управляющую конструкцию, например, unless (которая выполняет блок кода, если условие ложно), в LISP и Java. В LISP такая конструкция реализуется с использованием макросов, а в Java – через создание дополнительного класса и метода, что делает код менее выразительным и менее гибким. Ниже представлен код программы на языке LISP:

```
(defmacro unless (condition &body body)
  `(if (not ,condition)
     (progn ,@body)))
```

Использование макроса автоматически расширяет синтаксис языка и выглядит естественно при написании кода, так как его вызов ничем не отличается от вызова встроенных функций. Результат работы программы представлен на рис. 1.

Рис. 1. Результат работы программы на языке LISP

В Java нет встроенной поддержки макросов, поэтому добавление конструкции unless требует обходного пути. Одним из вариантов является создание утилитного класса со статическим методом, но это не выглядит так же лаконично и требует дополнительных усилий при каждом использовании. Эквивалентный код на языке Java:

```
public class ConditionalUtils {
   public static void unless(boolean condition,
Runnable action) {
      if (!condition) {
         action.run();
      }
   }
}
```

Вызов функции отличается по написанию от встроенных, что может вызвать затруднения в чтении кода у разработчика. Результат работы программы представлен ниже (рис. 2).

Рис. 2. Результат работы программы на языке Java

Также макросы в LISP идеально подходят для создания гибких и мощных инструментов логгирования. В отличие от традиционных функций, макросы позволяют захватывать и выводить не только значения выражений, но и текст исходного кода, что особенно полезно при отладке и отслеживании ошибок. Например, макрос логгирования может принимать выражение, автоматически выполнять его, а затем выводить текст самого выражения и его результат в формате отладочного сообщения. Это упрощает анализ выполнения кода, поскольку программист получает полную информацию о контексте, в котором была вызвана операция. Благодаря макросам логгирование можно встроить непосредственно в синтаксис LISP, избегая дублирования и упрощая процесс поддержки кода, что делает его особенно удобным в больших и сложных системах.

Таким образом, макросы в LISP представляют собой мощный инструмент для создания новых синтаксических форм, позволяя расширять язык под нужды конкретных задач. Их преимущества включают гибкость, метапрограммирование, автоматизацию и оптимизацию производительности, что делает их незаменимыми в областях, требующих высокой производительности и адаптивности. Эта способность к расширению языка, присущая LISP, остается уникальной даже среди современных языков программирования и делает его отличным выбором для исследований и создания сложных систем. Благодаря тому, что макросы позволяют управлять синтаксисом, программист может настраивать язык для решения конкретных задач, что невозможно в языках без макросов [2].

Список литературы

1. Graham P. On Lisp: advanced techniques for Common Lisp. Englewood Cliffs. N. J. : Prentice Hall, 1994. P. 117–196.

2. Graham P. ANSI Common LISP. Englewood Cliffs. N. J. : Prentice Hall, 1996. P. 235–258.

Информация об авторах

Кареев Андрей Андреевич, студент, Пензенский государственный университет.

Михалев Андрей Геннадьевич, кандидат технических наук, доцент, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 510.755

ВЛИЯНИЕ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ НА КОГНИТИВНЫЕ СПОСОБНОСТИ

М. Е. Киселёва[1], В. О. Мальцева[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1]250058kwert@mail.ru
[2]Valeryanka.04@mail.ru

Аннотация. Рассматривается важность логического программирования как инструмента для развитие критического мышления и логики в целом. Уделяется некоторое внимание появлению и становлению декларативного подхода программирования и логического программирования, как следствия. Приводятся примеры исследований и экспериментов по внедрению логического программирования в школьную программу.

Ключевые слова: логическое программирование, программное обеспечение, когнитивные способности, критическое мышление, логическое мышление, программирование, разработка

Для цитирования: Киселёва М. Е., Мальцева В. О. Влияние логического программирования на когнитивные способности // Вестник Пензенского государственного университета. 2024. № 4. С. 63–65.

Основной проблемой в сфере разработки программного обеспечения является создание и последующая поддержка сложных систем, например, экспертных систем. Эта проблема не теряет своей актуальности и сегодня, оставаясь в центре внимания специалистов в области IT. Требования пользователей касательно надежности, удобства и функциональности программного обеспечения постоянно увеличиваются и усложняются, что вынуждает разработчиков искать новые, более эффективные пути программирования. Одним из решений этой проблемы является разработка и применение более совершенных абстракций, т.е. механизмов, позволяющих программистам работать на более высоком уровне, игнорируя мелкие детали реализации и сосредотачиваясь только на основной задаче.

Логическое программирование, зародившееся в 1970-х гг., стало примером таких абстракций. Логическое программирование, корнями уходящее в математическую логику, представляет собой парадигму программирования, где программы формируются как набор логических предложений, описывающих отношения между данными. Эти предложения, написанные на языке, близком к формальному языку логики предикатов, интерпретируются системой логического программирования для вывода новых фактов и получения ответов на запросы. Одним из языков данной парадигмы программирования является Prolog, который до сих пор применяется в различных сферах, например, в искусственном интеллекте, обработке естественного языка и баз данных, тем самым иллюстрируя свою эффективность в решении нестандартных задач [1].

Ключевая абстракция, заложенная в основу логического программирования, – это логическая переменная, а также унификация, как основной механизм вычислений. Унификация устраняет множество деталей манипулирования данными, что значительно упрощает процесс разработки. Эта абстракция позволяет создавать программы более компактными и читаемыми, что делает их более доступными для понимания. Суть логического программирования можно сформулировать

следующим образом: вместо описания того, как решить задачу, программист описывает, что конкретно нужно решить.

Логическое программирование может быть плодотворно использовано практически в любой прикладной области. Однако оно имеет особую ценность в тех областях, которые имеют большое количество каких-либо ограничений или правил. В особенности, если данные ограничения и правила поступают из нескольких разных источниках или если они быстро меняются. К таким областям применения можно отнести: системы баз данных, логические электронные таблицы, системы интеграции данных, системы управления предприятием и многие другие, что подчеркивает универсальность логического программирования в современных условиях [1].

Логическое мышление, без которого не только тяжело стать востребованным специалистом в области IT, но и непросто жить в современном мире, предоставляет возможность тщательно анализировать и оценивать данные, что, в свою очередь, позволяет принимать обдуманные решения, основанные на проанализированных фактах. Оно помогает разделять сложные и большие задачи, которые в целом виде решить очень трудозатратно или почти невозможно, на более мелкие, управляемые элементы, после чего можно разработать конкретные шаги для решения этих подзадач. Это умение помогает не только в повседневной жизни, но и в профессиональной деятельности [2].

В свою очередь, логика является основой для формирования критического мышления. Трудно представить формирование критического мышления в отрыве от формирования логической культуры человека, так как логическая культура дает основу для последующего формирования критического мышления. Логическое и критическое мышление тесно взаимосвязаны между собой: невозможно развить одно, никак не повлияв на другое. Поэтому эти навыки очень ценятся в современном мире, благодаря им можно не только стать более востребованным специалистом в своей сфере, но и быть более логичным и менее подверженным дезинформации в повседневной жизни [2].

Изучение логического программирования оказывает положительное влияние на когнитивные способности в целом, а также на критическое мышление в частности. Благодаря строго определенным рамкам логического программирования и всего декларативного подхода в общем, логическое программирование задает структурированную среду для четкого формулирования сути проблемы и ее анализа. Это, в первую очередь, требует от разработчика ясной постановки самой задачи, что является первостепенным шагом в решении любой проблемы. Таким образом декларативный подход программирования вынуждает разработчиков более четко формулировать свои мысли, ясно видеть проблему и выявлять ее суть. Если в императивном подходе программирования необходимо описать, как решить поставленную задачу, то в декларативном необходимо описать, что конкретно нужно решить, т.е. найти самую сущность задачи. Данный подход предписывает специалистам раскладывать сложную проблему на более мелкие части, что, в свою очередь, положительно сказывается на логическом мышлении программиста [1]. Данный процесс формулировки и анализа проблемы развивает видение причинно-следственных связей, а также выявление противоречий и оценку достоверности информации. Эти навыки являются основными в критическом мышлении.

Кроме того, разработчик, совершенствуя свои навыки в логическом программировании, в какой-то момент начинает анализировать собственный процесс мышления и находить ошибки в своих же рассуждениях. Это представляет собой очень ценный навык в современной жизни, который может помочь не только в профессиональной среде, но и в повседневной жизни.

В свою очередь, критическое мышление обладает многогранностью и может развиваться через решение абсолютно разных задач. Например, практическое обучение чему-то новому, будь то творчество или изучение новой, до этого неизвестной человеку темы, анализ различной литературы или оценка достоверности какой-либо информации. Все это может повлиять на развитие критического мышления.

Логическое программирование также является инструментом для развития критического мышления. Оно требует от разработчика умения четко выражать свои мысли, анализировать задачу, находить основную проблему, вести логическую цепочку рассуждений, разбивать одну большую задачу на более мелкие, логически связные подзадачи. Кроме того, логическое программирование развивает и творческий подход к решению проблем, так как предписывает программисту мыслить нестандартно [3], т.е. разработчик должен находить новые и оригинальные подходы для решения нестандартных задач со множеством ограничений.

Уже проводились различные исследования по внедрению логического программирования в школы, а именно: в программу начальных классов. Одним из самых свежих таких исследований стало исследование «Logic Programming at Elementary School: Why, What and How Should We Teach Logic Programming to Children?» *Laura A. Cecchi, Jorge Rodriguez, Verónica Dahl*, которое в дословном переводе звучит как «Логическое программирование в начальной школе: зачем, что и как нужно преподавать детям в рамках логического программирования?» Это исследование опубликовано в книге «Prolog: The Next 50 Years» и обращает внимание на преимущества раннего обучения логическому программированию [4].

Помимо этого, был разработан проект по включению изучения языка логического программирования Prolog в программу обучения в средней школе в Болгарии. Этот проект был принят, и проводился соответствующий эксперимент с группой отобранных учащихся, что подтверждает растущий интерес к логическому программированию.

Интеграция логического программирования в школьные учебные программы может стать одним из инструментов для формирования у детей навыков критического и творческого мышления, которые являются важнейшими навыками в современной жизни (ценными не только в профессиональной деятельности, но и в повседневной жизни).

Таким образом, логическое программирование не теряет своей актуальности и по сей день, представляя собой инструмент для развития логики в целом, критического и творческого мышления, в частности, которые являются важнейшими и очень ценными навыками для любого специалиста в современном мире.

Список литературы

1. Genesereth M., Chaudhri V. K. Introduction to Logic Programming. Morgan publishers, 2020. P. 3–10. URL: typeset.io>papers

2. Vartiak L., Jaseckova G., Milan K. Logic as a Tool for Developing Critical Thinking // Rupkatha Journal on Interdisciplinary Studies in Humanities. 2023. Vol. 15, № 2. P. 1–12.

3. Sterling L. Logic Programming and Software Engineering – Implications for Software Design // The Knowledge Engineering Review. 1996. Vol. 11. 345 p.

4. Warren D. S., Dahl V., Eiter T. [et al.]. Prolog: The Next 50 Years. Springer, 2023. P. 131–153. URL: link.springer.com>Book

Информация об авторах

Киселёва Марина Евгеньевна, студентка, Пензенский государственный университет.

Мальцева Валерия Олеговна, студентка, Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.414.22

ПРОЕКТ ПРИЛОЖЕНИЯ «ПОТЕРЯННЫЕ ЖИВОТНЫЕ»

М. Е. Киселёва[1], В. О. Мальцева[2], Е. А. Дзюба[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1]250058kwert@mail.ru
[2]Valeryanka.04@mail.ru
[3]Dzyuba_ea@mail.ru

Аннотация. Представлены анализ и разработка требований для приложения «Потерянные животные». Освещаются ключевая проблема, на решение которой направлено данное приложение, основная цель, главные задачи. Рассматривается целевая аудитория данной платформы. Разбираются функциональные и нефункциональные требования к приложению. Разрабатываются и приводятся диаграмма вариантов использования, а также концептуальная карта.

Ключевые слова: разработка приложения, анализ требований, разработка требований, функциональные требования, платформа

Для цитирования: Киселёва М. Е., Мальцева В. О., Дзюба Е. А. Проект приложения «Потерянные животные» // Вестник Пензенского государственного университета. 2024. № 4. С. 66–69.

Приложение «Потерянные животные» нацелено на помощь людям в поиске их пропавших питомцев, посредством объединения владельцев питомцев, неравнодушных людей, волонтеров, а также ветеринаров. Оно представляет собой платформу, которая объединяет людей для достижения общей цели – помощи оказавшимся в беде животным. Приложение также поможет создать сообщество людей, вдохновленных общей целью – спасением попавших в беду питомцев. Сообщество сможет морально поддержать в трудный момент людей, оказавшихся в такой непростой ситуации, и, помимо этого, сможет организовать поиск питомца более эффективно, чем если бы это делал один лишь хозяин.

Был проведен «мозговой штурм» и составлена концептуальная карта [1] для проекта приложения «Потерянные животные» (рис. 1).

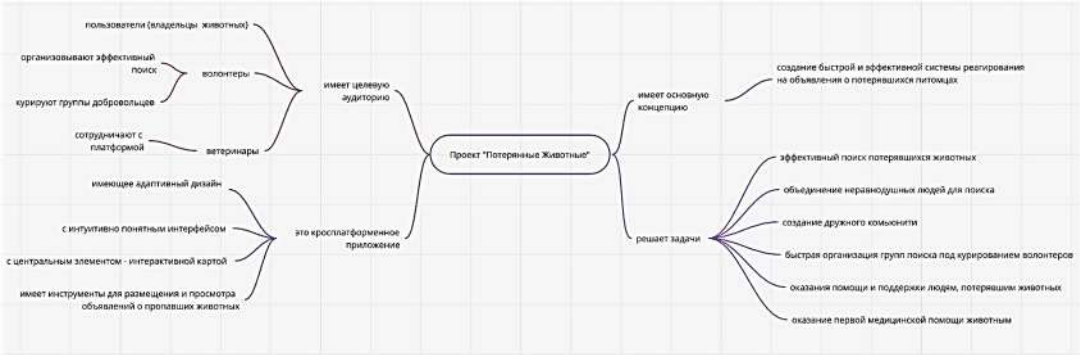


Рис. 1. Концептуальная карта проекта «Потерянные животные»

Концепция данного приложения заключается в быстрой и эффективной системе реагирования на объявления о потерявшихся животных. Ценность концепции заключается в следующем: в настоящее время люди, потерявшие любимца, что, к сожалению, не редкость, ищут питомца самостоятельно в близлежащем районе, но это малоэффективно. Данное приложение позволит создать связь между неравнодушными людьми и владельцами потерявшихся животных, что, в свою очередь, позволит, во-первых, расширить круг поиска и, во-вторых, приведет к построению эффективного плана поиска и конструктивной его организации. Также платформа сотрудничает с ветеринарными клиниками и приютами, связывая тем самым владельцев животных с ветеринарами для оказания первой медицинской помощи питомцу, в случае необходимости.

Целевой аудиторией приложения являются владельцы домашних животных, волонтеры, помогающие организовать поиск пропавших животных, ветеринарные клиники и питомники, которым будет выгодно размещать свои услуги в приложении для расширения клиентской базы и повышения осведомленности о своих услугах, а также неравнодушные люди, которые не являются волонтерами, но хотели бы ими стать или просто желают помогать братьям нашим меньшим. Таким образом, данное приложение станет мощным инструментом для эффективного поиска потерявшихся питомцев.

Основной целью данной платформы является объединение владельцев потерявшихся животных, волонтеров и ветеринаров для организации быстрого поиска питомцев. Ключевыми задачами являются: обеспечение инструмента для размещения и просматривания объявления о потерянных и найденных животных; предоставление инструмента коммуникации между пользователями для организации более продуктивного поиска; информирование о потерявшихся домашних животных в режиме реального времени, а также обеспечение функционала интерактивной карты, которая будет обеспечивать возможность быстрого добавления новой метки с точными координатами в месте, где заметили потерявшегося питомца.

В ходе анализа и разработки требований была составлена диаграмма вариантов использования [2] для данного приложения (рис. 2).

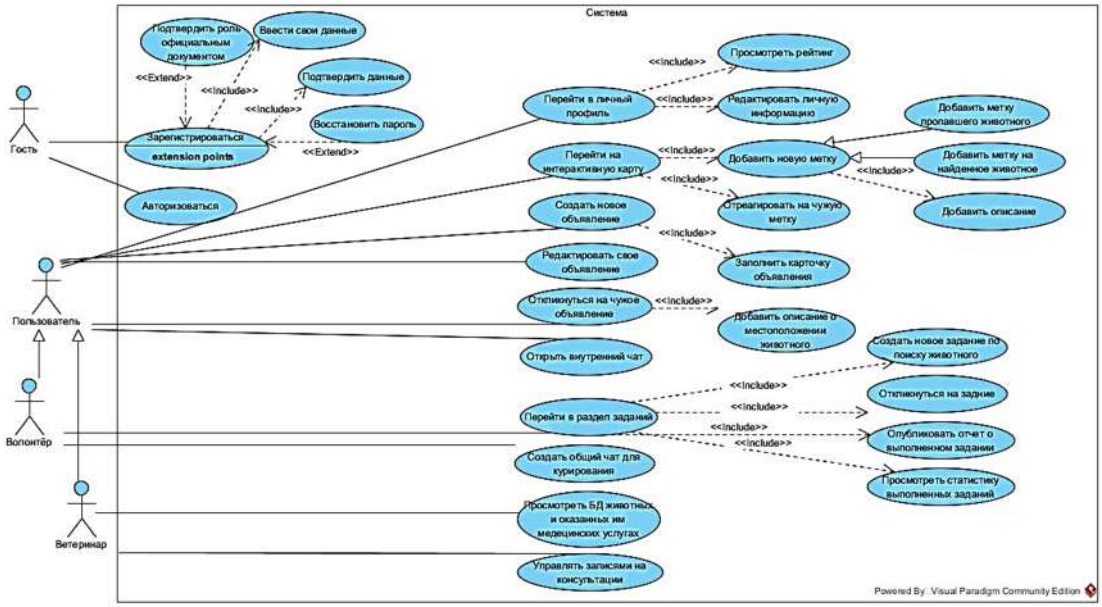


Рис. 2. Диаграмма вариантов использования для приложения «Потерянные животные»

Главным компонентом приложения является интерактивная карта, которая отображает места, где пользователи видели потерявшихся домашних животных. Пользователи смогут добавлять

метки, указывая точные координаты, где питомец потерялся или где питомца последний раз видели, а также прикреплять фотографии и описания. Данный функционал позволит более эффективно организовывать поисковые операции, так как волонтеры и владельцы домашних животных будут знать, на каких локациях в первую очередь следует сосредоточиться, а остальные пользователи будут уведомлены, если поблизости от них потерялся питомец. Также любой человек сможет буквально за пару минут отметить на карте точку, где был замечен потерявшийся питомец, что, в свою очередь, сможет очень сильно помочь в поиске и продвинуть его в нужное русло.

Рассмотрим функциональные требования к данной платформе [3]:

1. *Регистрация и вход*.

Приложение предусматривает возможность регистрации пользователей с разными ролями: пользователь, волонтер, ветеринар. Для получения двух последних ролей необходимо загрузить соответствующий документ для подтверждения.

Также при регистрации будет предусмотрена возможность подтверждения своих контактных данных. Вход в систему будет осуществляться при помощи логина и пароля. Помимо этого, будет предусмотрена возможность восстановления пароля.

2. *Интерфейс приложения*.

Интерфейс должен быть интуитивно понятным и удобным. Также должен быть предусмотрен адаптивный дизайн для разных мобильных устройств, мониторов ноутбуков и компьютеров, так как приложение разрабатывается как кроссплатформенное. Также должна быть предусмотрена функция переключения на разные языки.

3. *Основной функционал*.

Основные функции приложения включают в себя работу с объявлениями: создание объявление по шаблонизированной карточке (вид животного, фото, описание, контактная информация, метка на карте), редактирование и удаление собственных объявлений, возможность реагировать на объявления для указания актуальной информацией по животному, а также возможность добавления тэгов (порода, возраст) для облегчения поиска. Кроме того, предусматривается реализация функции «добавить в избранное», с помощью которой пользователи смогут быстрее находить нужное объявление.

Центральным элементом приложения является интерактивная карта, отображающая местоположения потерянных и найденных животных в виде меток, позволяющая настроить фильтры по расстоянию и типу животных (потерянные/найденные), с возможностью добавления новой метки, обозначающий местоположение потерявшегося питомца с указанием дополнительной информации, а также возможностью перехода к карточке объявления при нажатии на метку.

Помимо этого, платформа предусматривает возможность общения пользователей между собой при помощи встроенного мессенджера, а также функцией создания общих чатов для групп поддержки с целью координирования действий.

В приложении также предусматривается настройка уведомлений, включающих в себя: уведомления о новых потерянных животных в данном районе, уведомления об откликах на объявления и/или метки.

4. *Личный кабинет для разных ролей*.

Общий профиль отображает информацию о пользователе (имя, фото, контактные данные, роль) с возможностью редактирования.

Для волонтеров предусмотрен специальный раздел со списком текущих заданий. Для данной роли предусмотрен рейтинг, который повышается при выполнении заданий. Волонтеры могут создавать задания по поиску животного. Это нужно, например, если волонтер не может откликнуться на конкретное объявление, но он сможет создать задание, которое будут видеть все остальные волонтеры, и они смогут откликнуться на него. После выполнения задания они должны прикрепить отчет, после чего им начисляются баллы. Волонтеров с максимальным количеством

баллов можно будет просмотреть в рейтинге и в случае необходимости можно будет обратиться к ним за консультацией или помощью. Также предусмотрена статистика выполненных заданий.

Для ветеринаров и работников питомников разрешен доступ к информации о животных, принятых в приюты или клиники (медицинские данные, а именно: были ли проведены какие-либо операции или подобраны какие-либо препараты). Помимо этого, имеется функционал для управления записями на консультации (для ветеринаров), онлайн или офлайн.

Также у платформы будет администратор, для которого в личном кабинете предусмотрена панель администратора. К его возможностям относятся: управление пользователями, включая просмотр, редактирование, удаление профилей пользователей и блокировку пользователей за нарушение правил, модерация контента, а также аналитика и просмотр статистики.

К нефункциональным требованиям приложения относятся производительность, приложение должно обрабатывать не менее 1000 запросов одновременно без задержек, безопасность, включающая защиту личных данных пользователей и шифрование данных при передаче, стабильность, включающая минимальное время простоя приложения и регулярное резервное копирование данных, а также поддержка и обновления, т.е. обеспечение регулярных обновлений приложения и техническая поддержка для пользователей через FAQ и службу поддержки.

Таким образом, приложение «Потерянные животные» станет отличным инструментом для поиска и возвращения потерянных питомцев. Его четкая направленность, удобство и многофункциональность сделают его незаменимым. С помощью этого приложения будет построено тесное, дружелюбное сообщество, где каждый сможет внести свой вклад в спасение любимцев.

Список литературы

1. Kane M. A., Trochim W. Concept Mapping for Planning and Evaluation (Applied Social Research Methods). Thousand Oaks, Calif. : Sage Publications, 2006. 216 p.

2. Арлоу Д., Нейштадт А. UML2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. Символ-Плюс, 2007. 624 с. URL: litres.ru>UML2

3. Wiegers K., Beatty J. Software Requirements. Microsoft Press, 2014. 737 p. URL: books.google.ru

Информация об авторах

Киселёва Марина Евгеньевна, студентка, Пензенский государственный университет.

Мальцева Валерия Олеговна, студентка, Пензенский государственный университет.

Дзюба Елена Анатольевна, старший преподаватель кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.75

КЛАССИФИКАЦИЯ NK-ПОДОБНЫХ АВТОМАТОВ

Е. А. Кольчугина[1], В. А. Стежка[2]

[1]Пензенский государственный университет, Пенза, Россия
[2]АО «Радиозавод», Пенза, Россия

[1] kea@pnzgu.ru
[2] adevelgio@gmail.com

Аннотация. Представлено краткое описание NK-автомата Кауфмана. Приведены его отличительные характеристики: изменяемость и приспособленность. Описаны свойства, связанные с аттракторами. Проанализированы работы исследователей, изучающих NK-автоматы. Составлена классификация модификаций, в различной степени изменяющих базовые характеристики NK-автомата.

Ключевые слова: булевые сети, NK-автомат, аттрактор, степень эпистатичности, приспособленность автомата, модифицированный NK-автомат

Для цитирования: Кольчугина Е. А., Стежка В. А. Классификация NK-подобных автоматов // Вестник Пензенского государственного университета. 2024. № 4. С. 70–74.

Введение

В работе С. Кауфмана [1] для исследования процессов естественной самоорганизации и эволюции клеток живых организмов предлагается использовать модель NK-автомата. Она позволяет определить внутреннюю взаимосвязь элементов, описать порядок функционирования и спрогнозировать дальнейшее развитие с учетом ограничений, устанавливаемых самой моделью. N – это количество элементов в исследуемой модели, например, количество генов в генотипе или аминокислот в протеинах. K – это степень связанности элементов между собой. С точки зрения генетики, K – это степень эпистатичности. На рис. 1 представлен NK-автомат Кауфмана (A), в котором N = 4 и K = 2.

Процессы изменения (мутаций) в структуре NK-автомата приводят к появлению новых вариантов, для сравнения которых используется понятие «приспособленность» (fitness). Приспособленность f(x) NK-автомата — это среднее значение приспособленности каждого элемента.

$$f(x) = \frac{\sum_{i=1}^{N} f_i(x)}{N}, \quad (1)$$

где N – количество элементов; f(x), $f_i(x)$ – приспособленность всего автомата, каждого элемента; x – текущее состояние.

NK-автомат, как и все автономные бинарные сети, имеет конечный набор состояния своих элементов. И с учетом того, что сеть замкнута, неизбежно образуются циклические последовательности состояний, которые с течением времени повторяются. Такие циклы называются аттрактором, а последовательность состояний, приводящих к повторению, называется длиной аттрактора. Длина аттрактора варьируется от 1 до 2^N, причем чем меньше значение длины, тем более система устойчива к возмущениям, а долгое недостижение аттрактора интерпретируется как пе-

реход к хаосу. Аттракторы играют важную биологическую роль, так как они являются тем, что делает система. Аналогами аттракторов в биологическом смысле, например, являются различные типы клеток в организме. Для нейронной сети это память или категории, по которым сеть «узнает» окружающий мир.

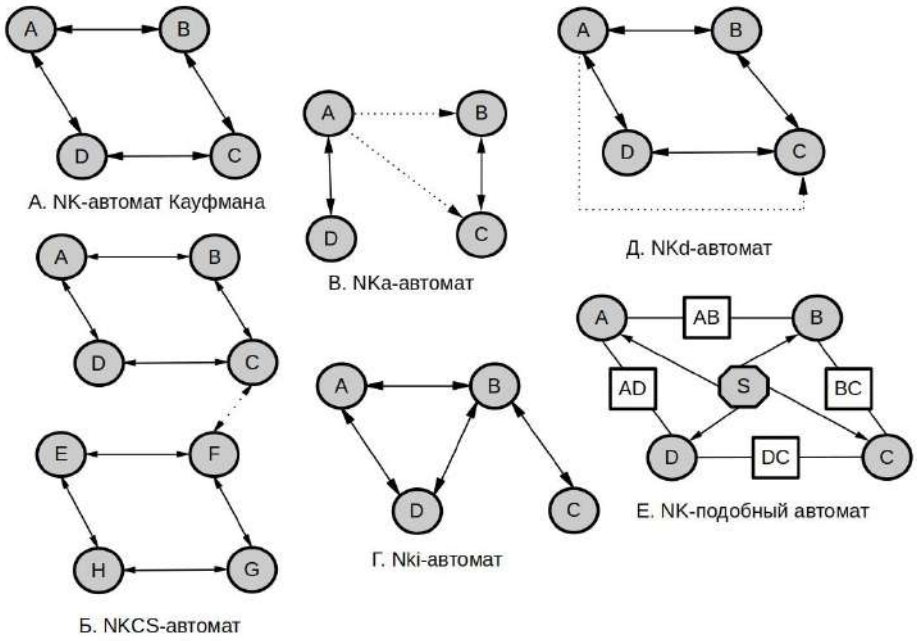


Рис. 1. NK-автоматы (NK-автомат Кауфмана (А), NKCS-автомат (Б), NKa-автомат (В), NKi-автомат (Г), NKd-автомат (Д), NK-подобный автомат (Е))

Базовый NK-автомат Кауфмана намеренно упрощен, благодаря этому в распоряжении исследователей появился простой, доступный и легко воспроизводимый аналитический инструмент. Как эволюция движется путем постепенного увеличения сложности, так и базовый NK-автомат в научном сообществе приобрел дополнительные свойства и особенности, в результате чего появился ряд модификаций, подражающих реальным системам.

Цель работы: провести обзор и классификацию модифицированных NK-автоматов, которые предлагается обобщить в следующие группы:

– блочные NK-автоматы; в структуре таких автоматов образуются обособленные независимые регионы, состоящие из подмножества элементов и связей между ними;

– комплементарные NK-автоматы; в таких автоматах степень эпистатичности элементов может быть различной;

– неравномерные NK-автоматы; вклад каждого элемента в общую приспособленность автомата может быть различным.

Предлагаемая классификация

Блочные NK-автоматы. Анализируя коэволюцию нескольких биологических видов, С. Кауфман и С. Йонсен предложили использовать NKCS-автомат [2], состоящий из нескольких (S) NK-автоматов, элементы которого (N), помимо внутренних связей (K), имеют и внешние связи (C). На рис. 1 показан NKCS-автомат (Б) с N = 8, K = 2, C = 2, S = 1. В общем случае процесс коэволюции связанных NK-автоматов имеет два состояния:

1) локальный оптимум одного «партнера» критически влияет на локальный оптимум связанных «партнеров», и система находится в постоянном изменении, продолжает «танцевать»;

2) система достигает устойчивого равновесия, в котором локальный оптимум каждого «партнера» согласуется с локальными оптимумами всех остальных «партнеров».

С течением времени поведение любого NKCS-автомата стремится к равновесию Нэша, и важной характеристикой становится время, за которое оно будет достигнуто. При K > SC система быстро достигает равновесия Нэша, а, если K < SC, система функционирует в хаотическом режиме, а равновесие становится трудно достижимым, причем, если значение K будет увеличиваться по отношению к C, система придет к уравновешенному состоянию быстрее.

RMNKCS-автоматы [3] являются результатом комбинации случайных булевых сетей (*RBN*) и NKCS-автоматов. Получившаяся модель позволяет исследовать взаимосвязь между фенотипическими признаками и сетью генетической регуляции, посредством которой они продуцируются. Например, возникающие мутации могут изменять логическую функцию случайного узла либо случайно изменять связи для этого узла. Изменение сети происходит до момента достижения аттрактора. На каждом шаге рассчитывается общая приспособленность, и, если получившееся значение выше, чем приспособленность другого вида, состояние становится родительским для последующих мутаций, в противном случае сеть возвращается к предыдущему состоянию. Таким образом у коэволюционирующих видов поддерживается максимум приспособленности.

NKa-автомат [4] добавляет параметр A, определяющий количество аффекторных элементов, которые будут влиять на всю модель. На рис. 1 показан NKa-автомат (В) с N = 4, K = A = 1, в котором выбран один аффектор и вклад каждого элемента в общую приспособленность зависит от внутреннего состояния и от состояния аффектора. Другое крайнее состояние A = N, в таком случае для каждого элемента выбирается K аффекторов из A. Особенность NKa-автомата заключается в том, что в модели будет не более 2^A локальных максимумов приспособленности, а по сравнению с базовым NK-автоматом количество шагов для достижения локального максимума уменьшается по мере увеличения числа аффекторных элементов. Кроме того, динамика уменьшения количества шагов оказывается более крутой для больших значений K.

Комплементарные NK-автоматы. Одно из упрощений базового NK-автомата заключалось в том, что все N элементов имеют одинаковую степень связанности K. Например, в цепочке ДНК вклад каждого локуса в общую приспособленность зависит от его аллелей и от K связей с другими локусами. Таким образом стоит предположить, что степень такого влияния для разных локусов может быть различной.

Для определения степени эпистатичности каждого локуса ДНК был предложен NKi-автомат [4] (см. рис. 1, NKi-автомат (Г) с N = 4, K = 2). Особенность NKi-автомата в том, что все NK связи распределены неравномерно, сосредоточены между некоторыми элементами. Например, для автомата с N = 14 и K = 4 число связей будет равно 56, а их распределение представлено в табл. 1.

Таблица 1

№ варианта	Количество связей элементов												
	1	2	3	4	5	6	7	8	10	11	12	13	14
1	0	0	0	0	0	14	14	14	0	0	0	0	0
2	0	0	0	0	8	10	10	10	8	0	0	0	0
3	0	0	0	7	7	7	7	7	7	7	0	0	0
4	4	4	4	4	4	4	4	4	4	4	4	4	4

Результаты работы семейства NKi-автоматов показывают, что чем более однородно распределение эпистатичности, тем выше ожидаемая приспособленность локального максимума.

В базовом NK-автомате реализуется схема глобального управления: изменения принимаются на основе анализа состояния всех N элементов. В NKd-автомате [5] обобщается подход, используемый в NKCS-автомате, добавляя распределенное управление, каждый элемент из N получает дополнительный параметр D, обозначающий степень связанности с другими элементами (см. рис. 1, NKd-автомат (Д) с N = 4, K = 2, D = 1). Решение о принятии очередного изменения принимается на основе приспособленности D + 1 связанных элементов. Следовательно, в классическом NK-автомате D = N – 1, а в NKCS автомате D равняется количеству элементов каждого из S блоков. NKD-автомат показывает оптимальные показатели общей приспособленности модели при D ≈ 0,8N независимо от количества N.

Неравномерные NK-автоматы. Так как модель NK-автомата связана с определением общей приспособленности системы, существуют модификации, меняющие характер ее расчета. Согласно модели, предложенной Кауфманом, приспособленность определяется по формуле (1). Для определения параметра w перейдем к виду

$$f(x) = \frac{f_1(x)+\ldots+f_N(x)}{N} = \frac{1}{N}f_1(x) + \ldots + \frac{1}{N}f_N(x). \quad (2)$$

Вклад каждого элемента в общую приспособленность зависит от одинакового параметра, найденного по формуле

$$w = \frac{1}{N}. \quad (3)$$

Таким образом базовый NK-автомат является частным случаем **NKw-автомата** [4], для которого $0 < w_i < 1$ и $w_i + \ldots w_N = 1$. Отличительные особенности появляются если сильно увеличить различие w между элементами, тогда элемент с наибольшим значением, называется «сильно взвешенным». В автомате с одним «сильно взвешенным» элементом более высокое эпистатическое взаимодействие приводит к более высокой приспособленности. В автомате с двумя независимыми «сильно взвешенными» элементам для любых значений K среднее значение приспособленности локального максимума также будет выше, чем у обычного NK-автомата.

Модель NK-автомата Кауфмана не предусматривает конкурентный доступ к элементам при одновременно выполняемых вычислениях и подразумевает параллельный режим функционирования (все элементы срабатывают одновременно). Для устранения вышеперечисленных недостатков Е. А. Кольчугина предложила модификацию – NK-подобный автомат [6]. В NK-автомат добавлен управляющий элемент (S), определяющий порядок вычисления функциональных элементов (см. рис. 1, NK-подобный автомат (Е), N = 8, K = 2). Все выходы управляющего элемента соединены со всеми функциональными элементами (A, B, C, D), таким образом, что в каждый момент времени происходит вычисление только тех элементов, для которых значение управляющего входа активно (равно «1»). Дополнительно в автомат добавляются элементы, играющие роль замедлителя, хранящие состояние связанных элементов в предыдущий момент времени (AB, BC, DC, AD), позволяющие разделить процесс вычисления состояния автомата на отдельные параллельные цепочки.

В статье М. Ю. Бабича и А. М. Бабича [7] рассматривается модифицированный NK-автомат как «основа аппарата моделирования функций сложных систем, чьи изменения состояний представимы в виде ориентированного графа». Модификация заключается в изменении свойств элементов, выражающихся логическими функциями «И» и «ИЛИ», в процессе функционирования автомата. Модификация несколько изменяла численные характеристики NK-автомата Кауфмана, поэтому для определения приблизительного равенства состояний использовался параметр U. В результате исследования автоматов со сменой состояния подтвердилось основное свойство: начиная с K = 3, наблюдалось резкое увеличение длины аттракторов, что имитирует возникнове-

ние периода хаоса. Также было подтверждено, что возрастание параметра U ожидаемо увеличивает длину циклов управления.

Заключение

Модели NK-автоматов находят свое применение не только в теории искусственной жизни, для исследования механизмов поддержания упорядоченного и устойчивого состояния (гомеостаза) цифровых организмов, но и в экономике для модификации бизнес-процессов в организации [8], в физике для исследования свойств разбавленных магнитных сплавов (спиновые стекла). Благодаря упрощениям, принятым в NK-автоматах, например, дискретность, бинарность и детерминизм [9], базовый NK-автомат привлекателен своей лаконичностью и простотой, позволяющей доступно продемонстрировать процессы спонтанной самоорганизации.

Список литературы

1. Kauffman S. A. The Origin of Order: Self-Organization and Selection in Evolution. New York : Oxford University Press, 1993. 710 p.

2. Kauffman S. A., Johnsen S. Coevolution of the Edge of Chaos: Coupled Fitness Landscapes, Poised States, and Coevolutionary Avalanches // Journal of Theoretical Biology. 1991. № 149. P. 467–505.

3. Bull L. Coevolving Boolean and Multi-Valued Regulatory Networks. Bristol, UK : Computer Science Research Centre. University of the West of England, 2023. 21 p.

4. Solow D., Burnetas A., Roeder T., Greenspan N. Evolutionary Consequences of Selected Locus-Specific Variations in Epistasis and Fitness Contribution in Kaufman's NK-model. Cleveland, USA : Case Western Reserve University, 2001. 20 p.

5. Bull L. Exploring Distributed Control with the NK Model. Bristol, UK : Computer Science Research Centre. University of the West of England, 2020. 17 p.

6. Кольчугина Е. А. Исследование свойств цифровых организмов с помощью NK-подобных автоматов // Вестник компьютерных и информационных технологий. 2011. № 11. С. 20–24.

7. Бабич М. Ю., Бабич А. М. Возможности модифицированного NK-автомата Кауфмана при имитации особого периода функционирования систем // Технические науки. Информатика, вычислительная техника. 2018. № 4 (48). С. 17–27.

8. Levinthal D. A. Adaptation on Rugged Landscapes // Management Science. 1997. Vol. 43, № 7. P. 934–950.

9. Sansom R. Ingenious Genes: How Gene Regulation Networks Evolve to Control Development. Cambridge, Massachusetts : The MIT Press, 2011. 128 p.

Информация об авторах

Кольчугина Елена Анатольевна, доктор технических наук, доцент, профессор кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Стежка Владимир Андреевич, начальник программного отдела, АО «Радиозавод» (г. Пенза).

Авторы заявляют об отсутствии конфликта интересов.

UDC 004.42

ARTIFICIAL CHEMISTRY MODELS: ARTIFICIAL THOUGHTS FOR ARTIFICIAL MIND

E. A. Kol'chugina

Penza State University, Penza, Russia

kea@pnzgu.ru

Abstract. We propose a new concept of artificial intelligence based on principles of artificial chemistry, where short programs representing artificial molecules can spontaneously react with each other forming new programs or complex computing structures. These structures representing source and results of a special kind of intellectual activity are seen as artificial thoughts. The structure of "artificial mind" following this concept and methods of interactions between programs are also proposed.

Keywords: spontaneous emergence of programs, artificial chemistry models, artificial mind, artificial brain, artificial thought

For citation: Kol'chugina E. A. Artificial chemistry Models: Artificial thoughts for Artificial Mind // Vestnik of Penza State University. 2024; 4: 75–78. (In Russ.).

Introduction and background

Artificial chemistry [1] is one of the eldest types of computer models that appeared at the dawn of computing technology: J. von Neumann's self-replicating cellular automata [2] are claimed to be the first example of models of this type. Currently, artificial chemistry models are known as a sub-branch of software models of artificial life [3].

The set of artificial chemistry models is not uniform; these models use different mathematical apparatus. A wide variety of such models are functionally oriented. Their base is λ-calculus and its analogues.

The first model of this kind, *AlChemy* [4], is one of the first models of artificial life that studied the formation of finite sets of functions with a steady composition capable of self-maintenance and self-reproduction.

In this artificial chemistry, functions play a dual role of both reagents (molecules) and rules for conducting reactions. During the simulation, the molecules collide. The collision can be elastic, without consequences, and reactive, in which new molecules are formed. The easiest and most obvious way to form a molecule is to create a composition of reagents.

This approach is popular in artificial chemistry, and it has been further developed in some models, for example, such as *Gamma* [5] and *CHAM* [6], where analogous of λ-calculus has been proposed. In comparison to *AlChemy*, these newer models are focused on practical use, namely the automation of programming and the creation of new programming paradigms.

These models generously apply the metaphor of chemistry, including notions of "ions", "heating", "cooling" and "freezing" [6]. The resulting programming concepts are allowing creation of programs with dynamically changing structure and composition, in which functions and their compositions are

considered as a disposable material. The expenditure of such material means the end of the program execution.

Programming concepts based on this approach have many advantages, such as implicit parallelism, compact size of the resulting programs, high expressiveness of the programming style, and new capabilities to achieve automatic program development. But the list of disadvantages is not empty either. And the first disadvantage that should be mentioned is the function-oriented programming style used by these concepts. Function-oriented programming concepts are less popular among IT professionals due to an unusual style of thinking that is more difficult for a human to assimilate.

Another disadvantage is the non-obvious application area of such programming concepts. Currently, only a few applications are known in which dynamically changing programs are crucial.

Collectively, these disadvantages discourage a wide range of IT professionals from using such concepts. But it is obvious that the future belongs to methods based on evolution and self-organization: these methods can eliminate the influence of the human factor and make the final products more flexible and high-quality. In general, the function-oriented approach in artificial chemistry models is flourishing, evidence of which can be found in [7, 8].

The author has spent almost twenty years trying to bridge the gap between imperative programming and the paradigm of artificial chemistry and bring a decentralized and spontaneous character to the models of dynamically changing programs. The results in [9, 10] represent artificial chemistry models there functions are accomplished as active entities (possibly, program agents). In [9], the functions are understood as reactions for which initial substances are needed. Actually, reactions are looking for a substrate in the model space. In [10], the functions correspond to artificial analogues of atoms capable of spontaneous formation of molecules. Some kinds of atoms are stationary, and some kinds of atoms are mobile (volatile) and are also looking for other atoms to react with.

The results of the experiments described in [9, 10] led to the emergence of dynamic computing structures in the form of programs with an implicit [9] or explicit construction [10]. These computing structures are aggregates of logical functions of different complexity, and thus they represent some logical decisions or problem solutions. These structures are also can be seen as material inscriptions of spontaneously arising ideas or thoughts of artificial mind, emerging from the interactions between logical elements.

This led to the idea of a new form of artificial intelligence based on the emergence of computing (logical) structures corresponding to principles similar to the function-oriented models of artificial chemistry. The structures themselves are artificial molecules, and the basic functional elements are artificial atoms.

The aim of this study is to propose and consider such an idea, as well to propose also a new form of artificial intelligence that can be used independently or in ensemble with other concepts, such as neural nets, agent-oriented models, decision trees, etc.

The method

Figure 1 represents the new concept of artificial intelligence named “artificial mind”, to be distinct from traditional concepts of artificial intelligence.

The main component of the diagram in Figure 1 is an artificial brain: a vessel in which spontaneous reactions occur and new molecules arise. The space inside the vessel can be cellular or have a different topology, Euclidean or non-Euclidean. The reactions proceed according to a set of reaction rules, which is also part of the artificial brain. In general, the idea of artificial brain is close to “magical stirring mechanism” from [6].

But it is necessary to mention that there is no central processing mechanism in this model: all atoms and molecules retain their own source of internal energy, hence reactions are equiprobable and spontaneous.

Traditionally, an internal energy means a processor time. This analogy is common in artificial chemistry models [1].

The crucial question is the construction or choice of the means to achieve selective asymmetric interactions between atoms and molecules.

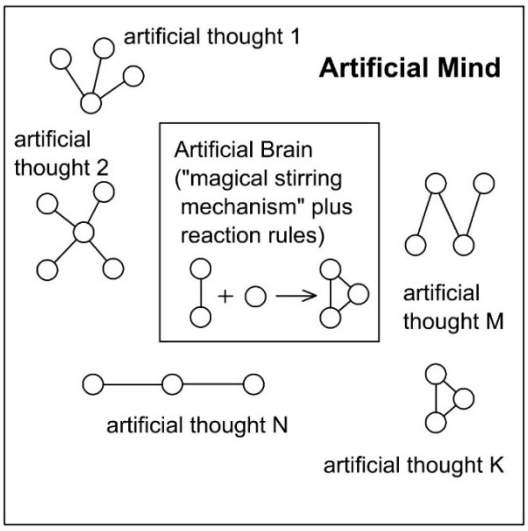


Fig. 1. The structure of artificial mind based on artificial chemistry approach

The analysis of various models of artificial chemistry [1–10] makes it possible to identify various kinds of such means. We state that the most powerful and perspective kinds are:

– connection by program labels [1];

– connection by common shared data structures [9];

– connection by specially designed tools imitating charged particles [10].

The latter two kinds of means provide decentralized spontaneous interactions between artificial atoms and molecules and the simulation of self-organization processes.

The initial and resulting molecules of the artificial intelligence model, or artificial thoughts, form an analog of the “solution” from [6]. The composition and structure of these molecules can be written in a graphical bracket notation from [10], which represents the forces of attraction and repulsion between artificial atoms and their strength.

These molecules are sources and results of activity of artificial mind itself or other kinds of artificial intelligence. For example, the atoms can represent primitive computing entities such as neurons in neural nets, transputers, or agents in agent-oriented models. Then the molecules correspond to neural nets with different topologies, transputer networks or self-assembling robots consisting of independent agents, software or hardware.

Results and discussion

We proposed a new concept of artificial intelligence and introduced the definitions of artificial mind, artificial thought and artificial brain.

Artificial thought is the material embodiment of a dynamic logical or computing structure consisting of dynamically interacting functions, just as a molecule consists of interacting atoms.

An artificial brain is a vessel in which artificial thoughts interact in a given model space in accordance with a set of reaction rules.

Artificial mind is a general notion for a new concept of artificial intelligence based on the principles of artificial chemistry and consisting of an artificial brain and artificial thoughts. Literally, in this concept, artificial thoughts interact with each other to form new artificial thoughts.

The advantages of the proposed concept are its decentralized nature and focus on the processes of emergence and self-organization. It also makes possible to store emerging dynamic structures as artificial thoughts using a form of graphic notation [10].

The disadvantages are related to the general problems of artificial chemistry models, for example, the limited capabilities of tools for organizing interactions between artificial atoms and molecules.

Conclusion and further studies

The proposed new concept of artificial intelligence can be used independently or together with other concepts and models, enabling the results of modeling or decision-making, such as emerging dynamic computing structures, to be stored in a tangible form. This can be used in the development of neural nets topologies, agent-based modeling, design of decision trees, etc. Future studies should be devoted to both improving the concept itself and expanding the possible scope of application.

References

1. Banzhaf W., Yamamoto L. Artificial Chemistries. Cambridge, Massachusetts; London, England: The MIT Press, 2015. 576 p.

2. Von Neumann J. Theory of self-replication automata / ed. by A. W. Burks. Urbana, London: University of Illinois Press, 1966. 416 p.

3. Artificial Life Models in Software / ed. by M. Komosinski, A. Adamatzky. London: Springer, 2009. 462 p.

4. Fontana W. Algorithmic Chemistry, Artificial Life II, SFI Studies in the Sciences of Complexity. Vol. X / ed. by C. G. Langton, C. Taylor, J. D. Farmer, S. Rasmussen. Redwood City, CA: Addison-Wesley, 1991. P. 159–209.

5. Banâtre J. P., Le Métayer D. The Gamma model and its discipline of programming // Science of Computer Programming. 1990. Vol. 15, № 1. P. 55–77.

6. Berry G., Boudol G. The Chemical Abstract Machine // Theoretical Computer Science. 1992. Vol. 96, № 1. P. 217–248.

7. Kruszewski G., Mikolov T. Emergence of self-reproducing metabolisms as recursive algorithms in an artificial chemistry// Artificial Life. 2021. Vol. 27, № 3–4. P. 277–299.

8. Cudennec L., Goubier T. A short overview of executing Γ chemical reactions over the ΣC and τC dataflow programming models // Procedia Computer Science. 2015. Vol. 51, Iss. C. P. 1413–1422.

9. Kol'chugina E. A. Spontaneous Emergence of Programs from «Primordial Soup» of Functions in Distributed Computer Systems // Automatic Control and Computer Sciences. 2018. Vol. 52, № 1. P. 40–48.

10. Kol'chugina E. A. Self-Synthesis of Programs Based on Artificial Chemistry Model // Programmnaya Ingeneria. 2022. Vol. 13, № 10. P. 440–448.

Information about the authors

Kol'chugina Elena Anatolyevna, Doctor of Technical Sciences, associate Professor, Professor of the department of mathematical support and application of electronic computing machines, Penza State University.

The author declare no conflicts of interests.

УДК 004.925.86

ЭКСПЕРИМЕНТАЛЬНОЕ ИССЛЕДОВАНИЕ ПАРАМЕТРИЧЕСКОЙ ИНТЕРПОЛЯЦИИ И ВИЗУАЛИЗАЦИИ НЕАНАЛИТИЧЕСКИХ КРИВЫХ

Ю. Н. Косников[1], Д. Р. Абубекеров[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1] kosnikov@gmail.com
[2] ofjordan@yandex.ru

Аннотация. Предложен интерполянт нового вида для реконструкции кривых, заданных множеством опорных точек. Интерполянт получен адаптацией интерполянта на основе радиальных базисных функций к зависимости от одного параметра. Смешивающие функции интерполянта действуют вдоль оси параметрической системы координат, эта ось совпадает с реконструируемой кривой. Центры смешивающих функций размещаются в опорных точках. Координаты промежуточных точек кривой вычисляются как суммы взвешенных значений смешивающих функций. Качество интерполяции зависит от многих факторов, в частности от вида смешивающих функций, числа опорных точек сегмента, участвующих в вычислениях и визуализации, и расстояний между опорными точками. Провести аналитическое исследование свойств предлагаемого интерполянта затруднительно. Описывается программа, позволяющая в интерактивном режиме определить влияние различных факторов на качество интерполяции. Программа выполнена на языке MatLab.

Ключевые слова: кривая, опорная точка, реконструкция, параметрическая интерполяция, визуализация, смешивающая функция, программа, точность интерполяции, гладкость кривой

Для цитирования: Косников Ю. Н., Абубекеров Д. Р. Экспериментальное исследование параметрической интерполяции и визуализации неаналитических кривых // Вестник Пензенского государственного университета. 2024. № 4. С. 79–83.

Криволинейные геометрические конструкции применяются при решении многих прикладных задач визуализации. С их помощью формируются изображения траекторий движения транспортных средств, отображаются результаты научного эксперимента, представляются границы земельных участков и геологических пластов, проводятся изолинии и границы зон распространения воздействий или физических величин. Один из вариантов задания формы кривых – множество характерных (опорных, контрольных) точек. В этом случае для реконструкции всей кривой требуется интерполяция. Методов интерполяции много, следовательно, при решении конкретной прикладной задачи возникает проблема выбора метода.

Хорошими формообразующими свойствами обладают интерполянты на основе смешивающих функций (СФ). Каждая опорная точка снабжается функцией одного или двух аргументов, которая в опорной точке получает экстремальное значение и плавно изменяется при отходе от опорной точки. Координаты промежуточных точек получаются путем суммирования (смешивания) взвешенных значений СФ. Весовые коэффициенты (коэффициенты влияния опорных точек) зависят от реконструируемой геометрической формы.

Для реконструкции кривых по опорным точкам широко применяются сплайны. Их смешивающими функциями, в частности, являются степенные полиномы различного вида. Аппарат сплайнов применяется десятки лет, но актуален и в настоящее время. Например, в публикации [1] 2022 г. описана интерполяция неаналитического контура с помощью B-сплайнов. Как правило, сплайны описываются в параметрической форме. При изменении параметра в заданном диапазоне в визуальной форме представляется отрезок между двумя опорными точками. Можно предложить интерполянт другого вида, у которого визуально представляемый отрезок соединяет несколько опорных точек – три, четыре и более. Применение такого интерполянта сокращает число отсеков составной кривой, упрощает алгоритм интерполяции и повышает ее производительность.

Предлагаемый интерполянт получается путем модификации интерполянта на основе радиальных базисных функций (РБФ). Такой интерполянт успешно применяется для моделирования гладких поверхностей, точно проходящих через множество опорных точек. Например, в публикации [2] описаны целый ряд РБФ и их применение для интерполяции. При параметрической форме записи РБФ-интерполянт является функцией двух переменных – параметров. Отсеки поверхностей, представляемые визуально при пробегании параметрами заданных интервалов, проходят через несколько опорных точек. Промежуточные точки отсеков становятся вершинами полигональной сетки, подлежащей визуализации средствами графической системы компьютера. Если перевести РБФ-интерполянт из трехмерного пространства на плоскость и ввести зависимость координат промежуточных точек от одного параметра, можно использовать интерполяционные свойства РБФ для реконструкции кривых.

Математическое описание предлагаемого интерполянта получено из РБФ-интерполянта устранением зависимости координат промежуточных точек от второго параметра. В этом случае возникает параметрическая система координат, имеющая криволинейную ось, которая проходит через опорные (и промежуточные) точки, т.е. совпадает с реконструируемой кривой. Опорные точки, кроме декартовых координат x_i, y_i, получают параметрическую координату t_i, где i – номер опорной точки. Для реконструкции кривой нужно найти декартовы координаты промежуточных точек каждого ее отсека. С этой целью организуется перебор параметрических координат в диапазоне от $t_{\min}$ до $t_{\max}$ и определение декартовых координат x, y по интерполяционным выражениям

$$x = \sum_{i=1}^{N} \lambda_{xi} \varphi(r_i), \tag{1}$$

$$y = \sum_{i=1}^{N} \lambda_{yi} \varphi(r_i), \tag{2}$$

$$r_i = |t - t_i|,$$

где N – количество опорных точек, оказывающих влияние на промежуточные точки; r_i – параметрическое расстояние между промежуточной (текущей) точкой и i-й опорной точкой; $\varphi(r_i)$ – значение смешивающей функции i-й опорной точки на удалении r_i от ее центра (ядра); λ_{xi}, λ_{yi} – коэффициенты влияния (весовые коэффициенты) i-й опорной точки на текущую точку.

Следует отметить, что добавление третьего аналогичного (1) и (2) уравнения для координаты z дает описание интерполянта пространственных кривых. Далее для простоты речь будет идти только о плоских кривых.

В вычислениях участвуют коэффициенты влияния опорных точек. Они должны быть найдены предварительно из условия точного прохождения кривой через опорные точки. Условие представляется в виде двух систем линейных алгебраических уравнений (СЛАУ) вида

$$\sum_{i=1}^{N} \lambda_{xi} \varphi(r_{ij}) = x_j, \tag{3}$$

$$\sum_{i=1}^{N} \lambda_{yi} \varphi(r_{ij}) = y_j\,,\ j = 1\,..\,N, \tag{4}$$

где r_{ij} – параметрическое расстояние между i-й и j-й опорными точками.

СЛАУ разрешаются относительно коэффициентов λ_{xi}, λ_{yi}, которые подставляются в (1), (2).

Количество опорных точек в реконструируемой кривой может быть весьма велико. Чтобы снизить размерность СЛАУ, в этом случае кривую составляют из сопрягающихся отсеков. Тогда выражения (1), (2) описывают точки одного отсека. Сопряжение отсеков достигается за счет двух приемов. Во-первых, при нахождении промежуточных точек каждого отсека в вычислениях участвуют некоторые точки соседних отсеков, т.е. отсеки частично перекрываются. Во-вторых, при формировании каждого отсека в визуальной форме представляются не все его промежуточные точки, а лишь ряд средних точек, образующих визуальный сегмент. Возникает схема расчета-визуализации вида $N \mapsto M$, где N – количество опорных точек, участвующих в вычислениях, а M – количество опорных точек, соединяемых в визуальном сегменте. Эти количества нуждаются в обосновании.

Характеристики качества интерполяции зависят от выбора схемы расчета-визуализации, расстояний между опорными точками и вида СФ. Провести аналитическое исследование свойств предлагаемого интерполянта по выражениям (1), (2) затруднительно. Выходом является экспериментальное исследование этих свойств. Предлагается программа, позволяющая в интерактивном режиме определить влияние различных факторов на качество интерполяции. Программа выполнена на языке MatLab.

Функционал программы включает следующие операции:

– визуализация тестовой функции в виде двумерной кривой, располагаемой в окне вывода. Для статистического подтверждения результатов интерполяции в качестве тестовой функции могут использоваться различные зависимости. Размер окна вывода может масштабироваться;

– выделение опорных точек тестовой кривой путем указания манипулятором «мышь». При этом координаты опорных точек в окне вывода определяются автоматически и заносятся в память. Применяется целочисленная последовательная параметризация;

– выбор вида смешивающей функции. Спектр СФ довольно широк.

В программе для определенности использованы:

1) радиальная базисная функция «инверсный мультиквадрик» [2]:

$$\varphi(r_i) = \frac{1}{\sqrt{1+\varepsilon(t-t_i)^2}},$$

где ε – коэффициент сглаженности (формы);

2) биквадратная координатно-линейная функция [3]:

$$\varphi(r_i) = (1 - {r_i}^2)^2,$$

где $r_i = \frac{t-t_i}{t_{\max}}$, а $t_{\max}$ – размер половины зоны влияния опорной точки в параметрических координатах;

3) бигармоническая функция Треффца (Trefftz) [4]:

$$\varphi(r_i) = 2r_i^2 \ln(r_i) + \left(1 - r_i^2\right),$$

при вычислении которой должно выполняться условие $\varphi(0) = 1$;

4) обратная функция Менандро (Menandro) третьей степени [5]:

$$\varphi(r_i) = -\frac{4}{1+r_i^3} + \frac{6}{1+{r_i}^2} - 1.$$

Все СФ задаются математическими формулами, в связи с чем в программу могут быть введены и другие разновидности СФ;

– задание схемы расчета-визуализации;

– вычисление коэффициентов влияния опорных точек путем решения СЛАУ вида (3), (4);

– вычисление промежуточных точек интерполянта по выражениям (1), (2);

– отображение результата интерполяции в окне вывода для визуального контроля формы;

– определение погрешности интерполяции по формуле среднеквадратического отклонения и метрике Хаусдорфа.

На рис. 1 показан вид окна вывода программы с тестовой кривой синего цвета, имеющей параметрическое описание вида (t – параметр):

$$x(t) = t - 2\sin(t) - 2,$$

$$y(t) = 1 - 2\cos(t) + 0.1t^2 - 5, \quad t = 1, 2, \dots, 11.$$

Кривая снабжена опорными точками.

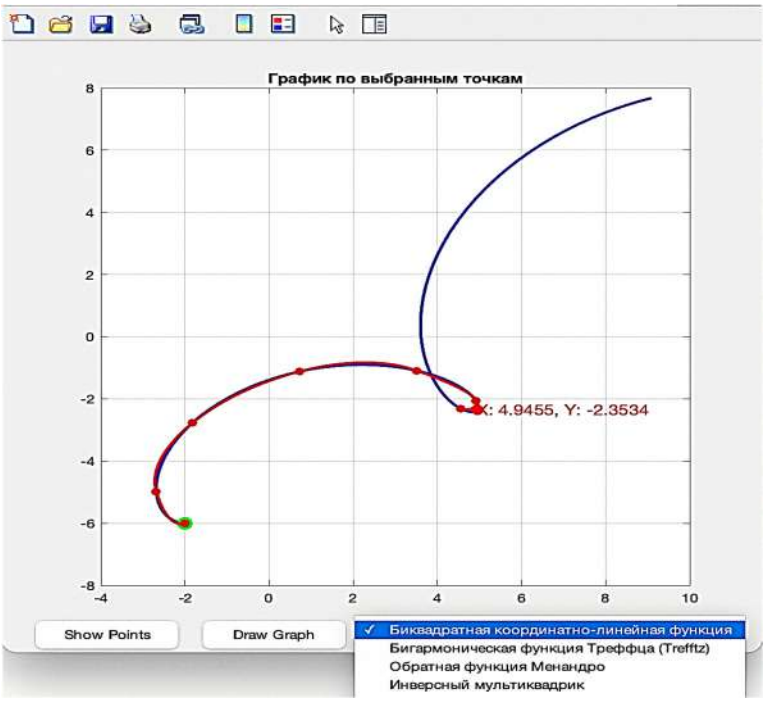


Рис. 1. Визуализация тестовой и интерполированной кривых

На данном рисунке показан сегмент интерполированной кривой красного цвета, построенный с применением биквадратной СФ. Использована схема расчета-визуализации вида 7↦5. Для наглядности изображения опорные точки расставлены настолько редко, чтобы возникло видимое расхождение кривых. На рис. 2 укрупненно показан участок наибольшего расхождения кривых – в области петли.

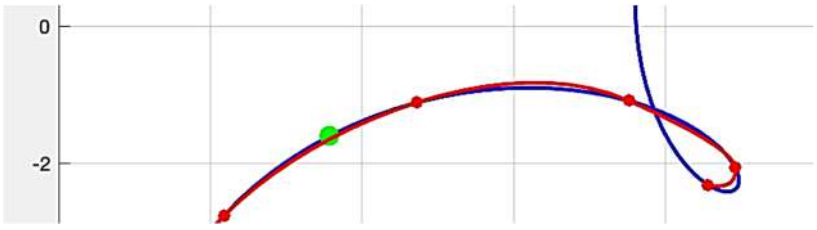


Рис. 2. Область наибольшего расхождения тестовой и интерполированной кривых

В этом случае среднеквадратическая погрешность интерполяции равна 6,78 %, а метрика Хаусдорфа дает значение 0,118.

Создан программный инструмент, который позволяет проводить визуальный и количественный анализ зависимости качества интерполяции от расстановки опорных точек, выбора СФ

и ее схемы расчета-визуализации. Использование программы позволит обоснованно выбирать параметры интерполяции для визуального представления плоских геометрических объектов произвольной формы. При возникновении задачи визуализации конкретного геометрического объекта к его фрагментам можно «примерить» различные близкие по форме аналитические кривые и применить различную расстановку опорных точек, различные СФ и различные схемы расчета-визуализации. Оценив варианты по погрешности интерполяции, можно осуществить обоснованный выбор.

Список литературы

1. Wei J., Sun C., Zhang Xj. [et al.]. An efficient and accurate interpolation method for parametric curve machining // Sci. Rep. 12. – 16000 (2022). URL: https://doi.org/10.1038/s41598-022-20018-9

2. Skala V., Mourycova E. Meshfree Interpolation of Multidimensional Time-Varying Scattered Data // Computers. 2023. № 12. P. 243. URL: https://doi.org/10.3390/computers12120243

3. Косников Ю. Н. Целочисленная параметризация геометрических форм на координатной плоскости // Международный научно-исследовательский журнал. 2023. № 12 (138). С. 1–9. URL: https://research-journal.org/archive/12-138-2023-december/10.23670/IRJ.2023.138.15

4. Piltner R. Exploring New Options for Data Interpolation with Radial Basis Functions // Workshop on Environmental Health and Air Pollution. 2018. P. 1–4. doi: 10.4108/eai.21-6-2018.2276666

5. Menandro F. Two new classes of compactly supported radial basis functions for approximation of discrete and continuous data // Engineering Reports. 2019. Vol 1. doi: 10.1002/eng2.12028

Информация об авторах

Косников Юрий Николаевич, доктор технических наук, профессор, профессор кафедры «Информационно-вычислительные системы», Пензенский государственный университет.

Абубекеров Динар Рашидович, аспирант, Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.432

СРАВНЕНИЕ ФУНКЦИОНАЛЬНОГО И ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Д. В. Крутяков[1], Г. Г. Кеделидзе[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1]krutyakov231@yandex.ru
[2]gioman5817@gmail.com

Аннотация. Проводится сравнительный анализ двух парадигм программирования – функционального и логического. Рассматриваются ключевые принципы каждой парадигмы, их сходства и различия, а также типичные примеры использования в современных языках программирования. Показаны основные преимущества и ограничения каждого подхода, что позволяет сделать выводы о наиболее эффективных сценариях использования данных парадигм.

Ключевые слова: функциональное программирование, логическое программирование, языки программирования, парадигмы программирования, сравнительный анализ

Для цитирования: Крутяков Д. В., Кеделидзе Г. Г. Сравнение функционального и логического программирования // Вестник Пензенского государственного университета. 2024. № 4. С. 84–86.

Целями данной статьи являются сравнение логической и функциональной парадигм программирования, определение факторов, влияющих на выбор между данными парадигмами при разработке конкретных информационных систем, а также выявление наиболее подходящей области применения для каждой из парадигм. Данная статья может помочь разработчикам программного обеспечения понять фундаментальные различия между функциональным и логическим программированием, что, в свою очередь, расширит их возможности для выбора наиболее подходящего инструмента для решения задач.

Для того чтобы сравнить функциональное и логическое программирование, были выбраны два классических представителя данных парадигм: Lisp и Prolog. Такие языки выбраны неслучайно: Lisp и Prolog были одними из первых и наиболее известных представителей функционального и логического программирования. Благодаря этим языкам появился базовый фундамент для этих парадигм, что, в свою очередь, послужило для дальнейшего развития схожих языков программирования [1]. Таким образом, изучив данные языки программирования и сравнив их, можно увидеть различия между логической и функциональной парадигмами на фундаментальном уровне.

Lisp – это один из старейших языков программирования высокого уровня, разработанный в 1958 г. Lisp был создан Джоном Маккарти для работы с искусственным интеллектом.

Название «Lisp» указывает на главную особенность этого языка. Это название – сокращение от LISt Processing language, что переводится как «язык обработки списков». В данном языке все «завязано» на списках. Каждый оператор – это список элементов, а каждый элемент может быть также списком или «атомом» (минимальной единицей данных).

Любому программисту известна такая условная конструкция, как if/else. Именно в Lisp впервые была реализована идея условных конструкций, а оттуда была позаимствована другими языками.

Lisp стал первым языком, который поддерживал рекурсию. Чтобы пройти по списку элементов, нужно использовать рекурсивную функцию, которая обрабатывает каждый элемент последовательно.

Реализация списков позволила осуществлять сохранение и очищение ячеек памяти динамически, за счет чего уже в первых вариациях языка появился сборщик мусора, один из основных элементов современных языков программирования.

Prolog – это один из самых известных языков логического программирования. Prolog благодаря своим особенностям используется в области искусственного интеллекта, компьютерной лингвистики и нечислового программирования в целом [2].

Особенность Prolog заключается в том, что пользователь, определяя факты и правила, которые описывают отношения между объектами, затем задает системе вопросы, на которые она должна ответить. Этот процесс осуществляется с использованием поиска решений путем унификации (сопоставления шаблонов) и логического вывода.

В качестве примера: в Prolog можно определить правило, описывающее родственные отношения, и затем задать вопрос: «Кто является родителем X?» Система сама проведет поиск по базе фактов и вернет соответствующие результаты. Такой пример программы изображен на рис. 1.

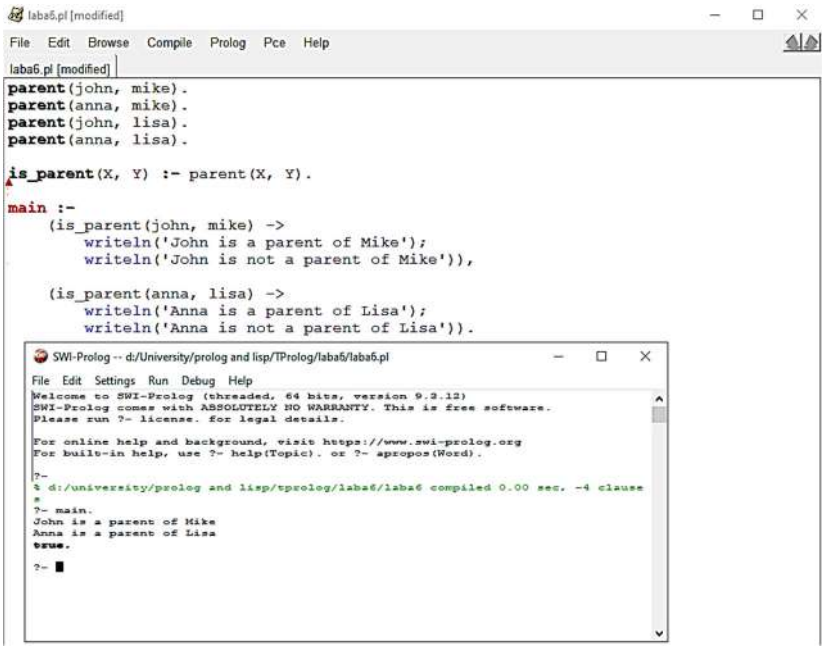


Рис. 1. Программа на языке Prolog

В свою очередь, программа с такой же задачей на языке Lisp показана на рис. 2.

Такой простой пример показывает, что в отличие от Prolog в Lisp требуется явно указать описание логики, что даже при небольшой программе делает код более сложным, тем самым усложняет время и выразительность задачи, связанные с логическими отношениями [3]. Таким образом, Prolog – более удобный язык для решения задач, связанных с представлением логических связей.

В Prolog не нужно описывать, как именно следует решить задачу (как в случае с функциями в Lisp). Это делает программы на Prolog очень выразительными, но одновременно и несколько сложными для понимания теми, кто привык к явным инструкциям.

```
(defparameter *parents* '((john mike) (anna mike) (john lisa) (anna lisa)))

(defun is-parent (x y)
  (member (list x y) *parents* :test #'equal))

(defun main ()
  ;; Sample queries
  (if (is-parent 'john 'mike)
      (format t "John is a parent of Mike~%")
      (format t "John is not a parent of Mike~%"))

  (if (is-parent 'anna 'lisa)
      (format t "Anna is a parent of Lisa~%")
      (format t "Anna is not a parent of Lisa~%")))

(main)
```

Рис. 2. Программа на языке Lisp

С другой стороны, необходимость написания полного алгоритма решения задачи делает Lisp более гибким языком программирования. Программист точно знает, как именно будет работать написанная им программа и в дальнейшем сможет легко изменить ее под свои нужды.

Также функциональное программирование в Lisp обладает таким преимуществом, как отсутствие побочных эффектов (чистые функции). При вызове функции выходной результат зависит только от входных параметров. Это позволяет средам выполнения программ заранее кешировать результаты функций и распараллеливать их без каких-либо действий со стороны программиста. Вышеупомянутые преимущества показывают, что при правильном написании программ на языках функционального программирования можно добиться высокой оптимизации и производительности.

Таким образом, в ходе сравнения логической и функциональной парадигм программирования было выяснено, что логическое программирование, в котором задача описывается декларативно, больше подходит для решения задач, связанных с поиском решений, экспертными системами и искусственным интеллектом. Функциональное программирование, в свою очередь, предполагает описание разработчиком конкретного алгоритма решения задачи, что может понадобиться при решении задач, подразумевающих научные вычисления и обработку данных.

Список литературы

1. Сергиевский Г. М., Волченков Н. Г. Функциональное и логическое программирование : учеб. пособие. М. : Академия, 2010. 320 с.

2. Prolog – удивительный язык программирования. URL: https://habr.com

3. Разница между прологом и лиспом. URL: https://ru.what-difference.com

Информация об авторах

Крутяков Даниил Вениаминович, студент, Пензенский государственный университет.

Кеделидзе Георгий Гочаевич, студент, Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.652

ОБЗОР РЫНКА IN-MEMORY СИСТЕМ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

А. А. Левин[1], С. А. Семичев[2], И. А. Казакова[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1] levin_andrej@vk.com
[2] semichev-serzh@mail.ru
[3] kia-2011@yandex.ru

Аннотация. Современные требования к электронным платформам обусловливают необходимость эффективного хранения данных, что делает системы управления базами данных критически важными для большинства приложений. Представлен обзор популярных In-Memory Databases, таких как Memcached, Redis, Apache Ignite, Tarantool и Oracle TimesTen In-Memory Database, с акцентом на их ключевые функции, применение и преимущества.

Ключевые слова: СУБД, IMDB, Memcached, Redis, Apache Ignite, Tarantool, Oracle TimesTen, эффективное хранение данных

Для цитирования: Левин А. А., Семичев С. А., Казакова И. А. Обзор рынка IN-MEMORY систем управления базами данных // Вестник Пензенского государственного университета. 2024. № 4. С. 87–89.

Реалии современного мира порождают высокий уровень требований к электронным платформам. Большинство приложений так или иначе нуждаются в хранении данных, начиная от информации о пользователях и их публикациях, заканчивая данными телеметрии, такими как файлы журналов и статистика использования. Наиболее распространенным и удобным средством для хранения подобной информации выступает база данных, к которой можно обращаться через систему управления базами данных (СУБД).

Системы управления базами данных представляют собой набор программных средств, которые обеспечивают контроль доступа к данным, хранящимся в базе, и управление их структурой. Эти системы выступают в роли посредника между пользователем и самой базой данных. Структура БД может различаться от реализации, но, как правило, это либо набор файлов, хранящихся на жестком диске или твердотельных накопителях, в которые СУБД сериализует информацию, либо набор структур данных, хранящихся напрямую в оперативной памяти. Такие СУБД называют In-Memory Databases (IMDB), и именно о них пойдет речь дальше [1].

IMDB имеет ряд отличий от классических баз данных, хранящих информацию на жестких дисках. В первую очередь это скорость. Доступ к данным в оперативной памяти значительно быстрее, чем к данным в файловой системе, так как отсутствует необходимость выполнения операций ввода-вывода через операционную систему, что уменьшает задержки. Кроме того, такие СУБД имеют упрощенную архитектуру, что позволяет снизить стоимость их использования. Однако объем хранимых данных, который значительно меньше, чем в классических СУБД, и долго-

вечность хранения данных, обусловленная энергозависимостью оперативной памяти, являются серьезными недостатками.

В силу своих преимуществ и недостатков In-Memory Databases получили широкое применение в качестве баз данных для кеширования, а не в качестве основной базы данных приложения. В индустрии очень распространен сценарий, при котором в качестве основной базы данных используется та, что хранит информацию в файлах (например, семейства SQL), а IMDB выступает хранилищем, помогающим снизить нагрузку на основную БД, путем хранения в себе самой используемой информации, которую можно быстро доставить [2].

Кратко рассмотрим наиболее популярные решения In-Memory Databases (IMDB):

1. Memcached. Это высокопроизводительная многопоточная система кеширования с открытым исходным кодом, появившаяся на свет в 2003 г. Она широко использовалась для ускорения веб-приложений, снижения нагрузки на базы данных и повышения масштабируемости. До появления множества аналогов, в которых было реализовано большее количество типов данных и ограничений, связанных с политикой вытеснения ключей, Memcached была стандартным инструментом кеширования [3].

2. Redis. Это высокопроизводительная система управления базами данных с открытым исходным кодом от компании Redis Labs, появившаяся в 2009 г. Первоначально являлась однопоточной системой, однако, начиная с версии 6, была реализована многопоточность. В начале развития проекта Redis имела одинаковые сферы применения с Memcached, однако за время существования проекта получила множество новых функций, что стало одним из факторов роста ее популярности. На сегодняшний день Redis – это выбор почти любой информационной системы, нуждающийся в IMDB [3].

3. Apache Ignite. Это распределенная система управления базами данных для обработки и вычисления значительных объемов данных в реальном времени с высокой эффективностью. Она основана на архитектуре клиент – сервер и может работать в кластерах, состоящих из нескольких серверов. Apache Software Foundation выпустила множество инструментов и приложений для разработчиков и обычных пользователей. Apache Ignite, выпущенная в 2015 г., – одна из таких систем, которая предоставляет возможности адаптивного хранения данных: помимо оперативной памяти, система также способна использовать постоянную память одного или нескольких узлов, а возможность настройки алгоритма распределения данных и поддержка стандартных SQL-запросов подчеркивают ее гибкость в использовании [4].

4. Tarantool. Это система управления базами данных с высокой производительностью, разработанная компанией VK и представленная в 2008 г. Tarantool совмещает в себе особенности нескольких различных СУБД и занимает промежуточную позицию между ними. Эта СУБД позволяет хранить различные типы информации, от таблиц до документов, поскольку у нее есть схемы, индексы и инструменты для шардирования; как и другие In-Memory СУБД, является хорошо масштабируемой [5].

5. Oracle TimesTen In-Memory Database. Это полнофункциональная реляционная база данных, разработанная компанией Oracle Corporation и представленная в 1997 г. Эта СУБД полностью поддерживает SQL семантику, работает на уровне приложений и сохраняет все данные в оперативной памяти. TimesTen обеспечивает все преимущества IMDB решений, а за счет записи данных и транзакций на диск устраняет существующие недостатки долговременности хранения данных [6].

На рис. 1 представлено сравнение наиболее важных характеристик рассмотренных СУБД.

Каждая из рассмотренных СУБД имеет свои преимущества и недостатки, а также общие качества, характерные для каждого из представленных вариантов. Однако однозначно выбрать СУБД для любого проекта невозможно, необходим тщательный анализ его требований.

	Memcached	Redis	Apache Ignite	Tarantool	Oracle TimesTen
Компания разработчик	Danga Interactive	Redis Labs	Apache Software Foundation	VK	Oracle Corporation
Лицензия	Open Source (BSD)	Open Source (Redis Source Available License v2)	Open Source (Apache License 2.0)	Open Source (BSD)	Проприетарная, закрытая
Открытый исходный код	Да	Да	Да	Да	Нет
Масштабирование	Вертикальное, горизонтальное	Вертикальное, горизонтальное	Вертикальное, горизонтальное	Вертикальное, горизонтальное	Вертикальное, горизонтальное
Возможность использовать в кластерах	Да	Да	Да	Да	Да
Тип базы данных	Ключ-значение	Ключ-значение	Мультимодальная	Мультимодальная, ключ-значение	Реляционная
Режимы хранения данных	Оперативная память	Оперативная память	Оперативная и постоянная память	Оперативная память	Оперативная память
Снепшоты	Нет	Да	Да	Да	Да
Транзакции	Нет	Да	Да	Да	Да

Рис. 1. Сравнение наиболее популярных In-Memory Databases

Список литературы

1. Кудашов А. С., Агапова В. А., Дьячков Д. А., Казакова И. А. Обзор типов индексов и их применение в системах управления базами данных // Современные цифровые технологии : материалы II Всерос. науч.-практ. конф. (г. Барнаул, 1 июня 2023 г.) / под общ. ред. А. А. Беушева, А. С. Авдеева, Е. Г. Боровцова, А. Г. Зрюмовой. Барнаул : Алтайский гос. техн. ун-т им. И. И. Ползунова, 2023. С. 299–303.

2. Что такое СУБД в оперативной памяти и как она эффективно сохраняет данные / Блог компании VK // Хабр. URL: https://habr.com/ru (дата обращения: 13.10.2024).

3. Разбираемся с Redis / Блог компании Wunder Fund // Хабр. URL: https://habr.com/ru (дата обращения: 14.10.2024).

4. Apache Ignite: как эта технология изменила подход к большим данным в Comindware / Блог компании Comindware // Хабр. URL: https://habr.com (дата обращения: 15.10.2024).

5. Tarantool: Билли Миллиган в мире СУБД / Блог компании VK // Хабр. URL: https://habr.com/ru (дата обращения: 16.10.2024).

6. Oracle TimesTen In-Memory Database // Oracle. URL: https://www.oracle.com (дата обращения: 17.10.2024).

Информация об авторах

Левин Андрей Алексеевич, студент, Пензенский государственный университет.

Семичев Сергей Алексеевич, студент, Пензенский государственный университет.

Казакова Ирина Анатольевна, доцент, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.6

АНАЛИЗ ВЛИЯНИЯ КАЧЕСТВА ДАННЫХ НА ПРОИЗВОДИТЕЛЬНОСТЬ АЛГОРИТМОВ МАШИННОГО ОБУЧЕНИЯ

А. Р. Майорова[1], Н. С. Карамышева[2], С. А. Зинкин[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1] oksaalla@yandex.ru
[2] karamyshevans@yandex.ru
[3] zsa49@yandex.ru

Аннотация. Исследуется влияние степени обработки данных на алгоритмы машинного обучения. Представлены результаты экспериментов, направленных на оценку влияния различных методов обработки данных на производительность различных моделей машинного обучения. Исследование производилось на одном и том же наборе данных, но с различными методами предварительной обработки.

Ключевые слова: машинное обучение, обработка данных, оптимизация моделей

Для цитирования: Майорова А. Р., Карамышева Н. С., Зинкин С. А. Анализ влияния качества данных на производительность алгоритмов машинного обучения // Вестник Пензенского государственного университета. 2024. № 4. С. 90–93.

В настоящее время существуют два подхода к увеличению точности машинного обучения: ориентированный на данные и ориентированный на улучшение параметров модели.

В современном машинном обучении данные играют ключевую роль, но их часто недооценивают или используют неправильно, в результате чего изменение параметров моделей на основе неподготовленных данных может занять продолжительное время для достижения адекватных метрик оценки [1].

В данной статье рассмотрено применение подхода увеличения точности модели, основанное на данных.

Для исследования был взят набор данных об успеваемости студентов вуза.

Представленный датасет содержит ряд параметров, влияющих на успеваемость студентов, разделенных на пять целевых групп.

Описание данных набора представлено в табл. 1.

Таблица 1

Описание данных

Атрибут	Описание
1	2
Sex	Пол
Age	Возраст
Weekly_study_hours	Время на внеурочную деятельность

Окончание табл. 1

1	2
Fails	Количество задолженностей
Job	Признак трудоустроенности студента
Familyedu	Признак, показывающий, имеют ли высшее образование оба родителя в семье
Activities	Признаки, отвечающие за дополнительные активности студента
Free_time	Количество свободного времени
Health	Рейтинг здоровья
Child_in_family	Количество детей в семье
Placeofbirth	Место рождения
Score	Средний балл по профильным предметам (учитывая отсутствие оценки как отсутствие предмета)
Predict	Целевая переменная

Для набора данных была оценена адекватность статистических показателей, построены матрицы корреляций значений. По результатам оценки данные были определены как пригодные для обучения моделей.

Эксперименты проводились с использованием пяти методов машинного обучения:

– логистическая регрессия;

– метод опорных векторов;

– случайный лес;

– метод *k* ближайших соседей;

– MLP (многослойный персептрон).

Обучение моделей было проведено для трех форматов данных:

– данных с минимальной обработкой: на данном наборе было применено кодирование признаков, удаления дубликатов и пропущенных значений;

– данных с полной обработкой: в обработку набора входило удаление аномалий, нормализация, балансировка классов; для балансировки был выбран метод SMOTE; данный алгоритм генерирует синтетические данные минорного класса;

– набора, содержащего только значимые признаки: выделение значимых признаков производилось с комбинацией критерия Хи-квадрат и метода Lasso.

Для кодирования категориальных признаков были использованы:

– Label encoding;

– one-hot encoding.

Настройки параметров моделей не изменялись в зависимости от набора. Разделение на тестовую и тренировочную выборки также не корректировалось.

Для метода *k* ближайших соседей определялся оптимальный *k* параметр для каждого эксперимента.

На первом этапе эксперимента была проведена сравнительная характеристика результатов обучения на обработанных данных с выделением значимых признаков и без их выделения.

Сравнение точностей моделей на разных наборах данных представлено в табл. 2. При использовании многослойного персептрона в метриках указано количество объектов тестового набора, которые были классифицированы неверно.

Из данной таблицы видно, что после балансировки классов несколько моделей демонстрируют рост точности. Это связано с тем, что разные модели по-разному реагируют на балансировку классов. Для балансировки был применен метод SMOTE, добавляющий синтетические данные к несбалансированным классам [2, 3], что не всегда оптимально, так как данный метод может приводить к переобучению. Но несмотря на это практически все модели продемонстрировали повышение точности в результате первого эксперимента.

Таблица 2

Результаты обучения на наборе данных для первого эксперимента

Модель	Обучение на данных без обработки	Обучение после балансировки классов	Обучение на данных после обработки и выделения значимых признаков
Логистическая регрессия	0,690	0,819	0,806
Метод опорных векторов	0,888	0,877	0,877
Случайный лес	0,985	0,989	0,994
k ближайших соседей	0,643	0,709	0,719
MLP	0,95(20)	0,96(16)	0,96(7)

В модели логистической регрессии балансировка повлияла на свободный член, что способствовало повышению точности. Метод опорных векторов не показал значительных улучшений в ходе эксперимента, но все же демонстрирует один из лучших результатов на используемом наборе.

В нашей выборке данных дисбаланс классов можно объяснить естественным распределением в реальных данных. Результаты обучения на несбалансированных данных похожи на те, что были получены при балансировке. Однако без балансировки некоторые модели склонны делать предсказания на основании класса большинства, что является ошибочным подходом. Также оценка производительности модели метрикой точности при несбалансированных классах может давать не совсем корректные показатели.

После выделения значимых признаков большинство моделей показали незначительное улучшение, за исключением логистической регрессии, в которой точность значительно снизилась, что говорит о том, что алгоритму не подошли выбранные признаки.

В методе опорных векторов результаты оставались неизменными, но уменьшение количества признаков способствовало ускорению модели. Также это может положительно сказаться на дальнейшей реализации.

Во втором эксперименте, чтобы повысить производительность некоторых моделей, выборка была разделена на три класса, были переопределены три новые группы.

Проведены классификация и анализ точности при делении на три класса, для данных со сбалансированными классами и с выделением значащих признаков. Результаты представлены в табл. 3.

Таблица 3

Результаты обучения низко результативных алгоритмов эксперимента

Модель	Обучение на данных со сбалансированными классами	Обучение на данных после обработки и выделения значимых признаков
Логистическая регрессия	0,950	0,915
Метод опорных векторов	0,952	0,925
K ближайших соседей	0,856	0,838

Из табл. 3 видно, что наблюдается увеличение показателей критерия сравнения. Результаты моделей значительно улучшились после перекластеризации. Улучшения наиболее выражены в моделях *k* ближайших соседей и логистической регрессии.

В модели логистической регрессии на результат повлияло то, что данный алгоритм является бинарным и при использовании многоклассовой классификации обобщается на несколько классов.

Для многоклассовой классификации применялись такие методы, как Classifier Chain и Multi Output Classifier. Метод Classifier Chain показал результаты лучше, чем Multi Output Classifier. Для оценки был использован наивысший показатель.

Метод k ближайших соседей не имеет ограничений по количеству классов, но при уменьшении их количества можно исправить проблему несогласованности и улучшить точность классификации.

В данной статье были рассмотрены результаты пяти алгоритмов машинного обучения, на основе данных разной степени подготовки. Также был проведен анализ влияния количества классов на эффективность моделей, что позволило выявить, как изменение числа классов может сказаться на точности алгоритмов.

Список литературы

1. Data-centric Machine Learning: Making customized ML solutions production-ready. URL: https://dida.do/blog/data-centric-machine-learning (дата обращения: 15.09.2024).

2. Боровская Е. В., Давыдова Н. А. Основы искусственного интеллекта. 4-е изд. М. : Лаборатория знаний, 2020. 130 с.

3. Губин Е. И. Методика подготовки больших данных для прогнозного анализа // Наука и бизнес: Пути развития. 2020. № 3 (105). С. 27–31.

Информация об авторах

Майорова Алла Романовна, аспирант, Пензенский государственный университет.

Карамышева Надежда Сергеевна, кандидат технических наук, доцент кафедры «Вычислительная техника», Пензенский государственный университет.

Зинкин Сергей Александрович, доктор технических наук, профессор, профессор кафедры «Вычислительная техника», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.5

ВИЗУАЛИЗАЦИЯ СУЩНОСТЕЙ ЛОГИСТИЧЕСКОГО СЕРВИСА: ОТОБРАЖЕНИЕ РЕЗУЛЬТАТА РЕШЕНИЯ ЗАДАЧИ МАРШРУТИЗАЦИИ В ИНТЕРФЕЙСЕ

М. В. Медведев[1], И. И. Антонов[2], И. Ю. Балашова[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1] schubert.walk@yandex.ru
[2] ilya.anton0v@vk.com
[3] irs-80@mail.ru

Аннотация. Рассмотрены проблемы визуализации данных маршрутизации в интерфейсе логистического сервиса. Проведен анализ алгоритмов маршрутизации с учетом ограниченного числа точек доставки. Оптимальными для задачи признаны алгоритмы ближайшего соседа и жадного поиска, обеспечивающие баланс между точностью и скоростью выполнения. Исследованы способы визуализации данных с использованием таблиц и графов. На основе анализа предложено комбинированное решение, включающее таблицы для управления данными и графы для наглядного представления маршрутов, что позволяет повысить удобство и эффективность работы пользователей с системой.

Ключевые слова: логистический сервис, визуализация данных, интерфейс, маршрутизация, алгоритмы маршрутизации, модель данных, таблицы, графы

Для цитирования: Медведев М. В., Антонов И. И., Балашова И. Ю. Визуализация сущностей логистического сервиса: отображение результата решения задачи маршрутизации в интерфейсе // Вестник Пензенского государственного университета. 2024. № 4. С. 94–97.

Рынок электронной коммерции активно развивается, что способствует увеличению объема грузоперевозок [1]. С ростом спроса на быструю и надежную доставку товаров возрастает потребность в специализированных логистических сервисах.

Одной из основных задач транспортной логистики является задача построения оптимального маршрута, минимизирующего затраты на транспортировку грузов [2]. Качественная визуализация маршрута улучшает понимание цепочки поставок и повышает уровень контроля над процессом доставки. Однако создание интерфейса логистического сервиса связано с рядом сложностей: нужно обрабатывать большие объемы данных, представлять их понятно и обеспечивать интерактивность, чтобы пользователь мог оперативно корректировать маршруты. Таким образом, актуальной задачей является разработка качественных интерфейсов, отображающих сущности логистического сервиса и результаты маршрутизации.

Одной из распространенных схем доставки является доставка товаров от склада, на который поступают товары от нескольких поставщиков, до нескольких торговых точек. Разработан логистический сервис, реализующий данную схему. На рис. 1 представлена концептуальная модель базы данных логистического сервиса.

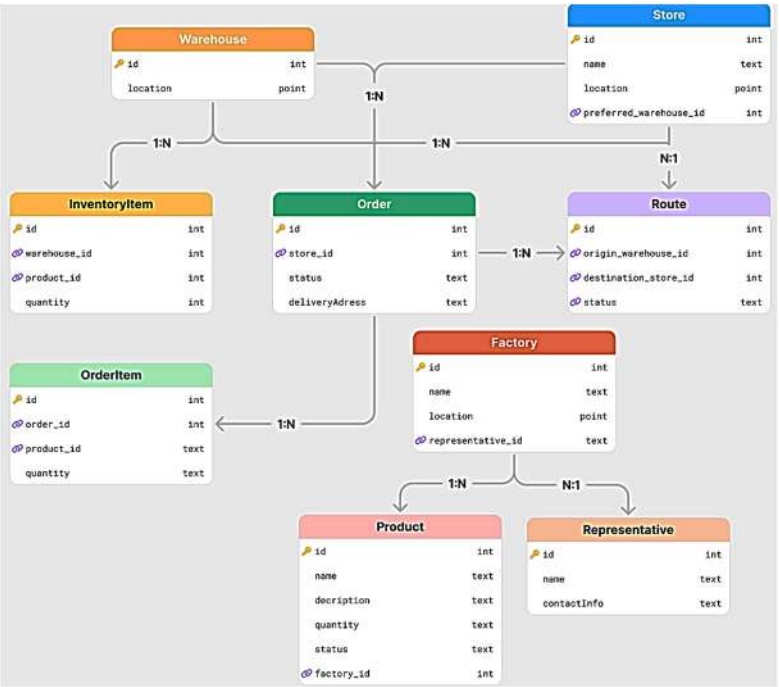


Рис. 1. Концептуальная модель базы данных

Основные сущности модели данных включают склад, завод, магазин, товар, заказ и маршрут. Склад хранит данные о запасах, завод отвечает за создание товаров, магазин размещает заказы, а маршрут оптимизирует путь доставки товаров.

В разработанном логистическом сервисе маршруты строятся на основе взвешенного графа транспортной сети, где вершины представляют склады и магазины, а ребра – пути между ними, каждое ребро имеет вес, например, расстояние. Для рассматриваемой системы, где число складов и магазинов невелико (до 15 узлов), эффективными будут алгоритмы оптимизации маршрута, не требующие значительных ресурсов для обработки данных, но обеспечивающие высокую скорость вычислений и адаптацию к изменениям. Основным критерием выбора алгоритма в этой задаче является скорость выполнения. Алгоритм должен быстро находить оптимальный маршрут, адаптируясь к изменениям, например, добавлению новых заказов, и быть устойчивым при малом числе узлов. Проведен анализ ряда алгоритмов оптимизации маршрута, получены следующие результаты:

1. Алгоритм ближайшего соседа и жадный алгоритм обеспечивают высокую скорость выполнения, легко адаптируются к изменениям и требуют минимальных ресурсов. Это делает их подходящими для маршрутов с ограниченным числом узлов. Однако они менее точны при поиске глобально оптимальных решений, чем генетический алгоритм и алгоритм ветвей и границ, что может стать критичным в сложных условиях маршрутизации.

2. Генетический алгоритм обеспечивает высокую оптимальность маршрутов и хорошую адаптацию к изменениям. Однако высокая вычислительная сложность и потребность в значительных ресурсах делают его менее подходящим для данной системы с небольшим количеством узлов.

3. Алгоритм ветвей и границ обеспечивает максимальную точность маршрута, но его вычислительная ресурсоемкость снижает эффективность для малых сетей, что делает его неоптимальным выбором в данных условиях.

Таким образом, для поставленной задачи наиболее подходят алгоритм ближайшего соседа и жадный алгоритм, поскольку они в случае схемы с небольшим числом магазинов обеспечивают наиболее быструю маршрутизацию и простую интеграцию в систему [3].

Для применения результатов решения задачи маршрутизации в системе недостаточно лишь оптимального построения маршрутов, также необходима качественная визуализация данных,

чтобы пользователи могли оперативно взаимодействовать с системой. Интерфейс должен позволять представителям складов и магазинов, а также администраторам легко отслеживать статусы поставок, контролировать маршруты и принимать решения в режиме реального времени [4]. Ниже приведены ключевые критерии выбора способа визуализации данных и сравнительный анализ двух основных подходов – таблиц и графов:

1. Наглядность статуса и прогресса – отображение текущих статусов заказов и этапов маршрутов.

2. Гибкость отображения – возможность переключения между обобщенной и детальной информацией.

3. Простота восприятия – отсутствие перегрузки при большом объеме данных.

4. Интерактивность – возможность корректировать маршруты.

5. Визуальная емкость – способность отображать большие объемы информации в удобном для восприятия формате.

Оценки каждого критерия выставлены по шкале от 0 до 2, где 0 означает, что способ визуализации не соответствует критерию, 1 – частично соответствует, а 2 – полностью удовлетворяет данному критерию. Результаты представлены в табл. 1.

Таблица 1

Сравнение представления данных в интерфейсе

Критерий	Таблица	Граф
Наглядность статуса и прогресса	1	2
Гибкость отображения	2	1
Простота восприятия	2	1
Интерактивность	1	2
Визуальная емкость	1	2

Как видно из данной таблицы, каждая из форм представления данных обладает своими сильными сторонами – таблицы эффективно отображают детализированную информацию о заказах и статусах, в то время как графы наглядно представляют маршруты на карте. Таким образом, для оптимального восприятия данных в интерфейсе логистического сервиса целесообразно использовать комбинированное решение – таблицы для отображения статусов и заказов, а графы для наглядного представления маршрутов на карте. Пример такого интерфейса представлен на рис. 2.

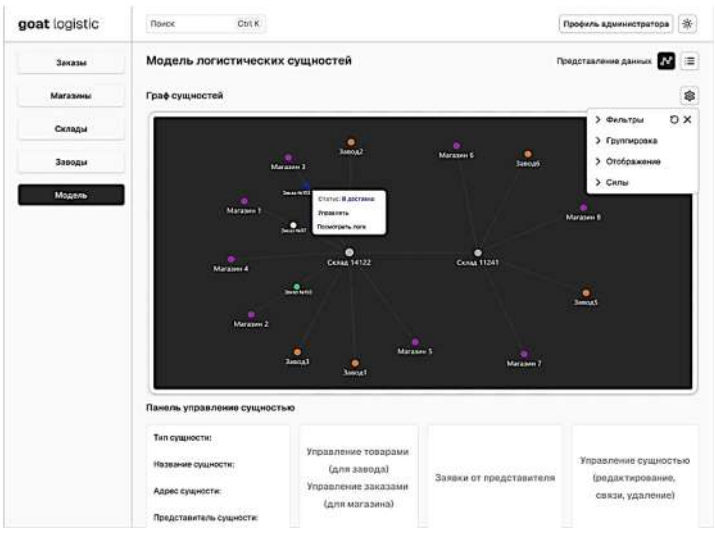


Рис. 2. Визуализация сущностей логистического сервиса

Предложенная архитектура и подходы к визуализации данных логистического сервиса позволяют повысить его удобство и эффективность, обеспечивая пользователям интуитивный и наглядный контроль над всеми этапами логистических процессов.

Список литературы

1. Тенденции грузоперевозок: поддержка государства, рост е-ком, цифровизация / РБК Компании. URL: https://companies.rbc.ru

2. Prasetyo A. Optimization in Transportation and Logistics: Enhancing Efficiency and Cost-effectiveness // Global Journal of Technology and Optimization. 2023. Vol. 14, № 2. P. 326.

3. Акмашев С. В., Балашова И. Ю. Сравнительный анализ стохастических и градиентных подходов к решению задачи оптимизации маршрутов транспортной логистики // Новые информационные технологии и системы (НИТиС-2023) : сб. науч. ст. по материалам XX Междунар. науч.-техн. конф., посвящ. 80-летнему юбилею Пензенского государственного университета. Пенза, 2023. С. 87–91.

4. Chen M., Yu S. Interactive Data Visualization for Decision-Making in Logistics Systems // Journal of Logistics Management. 2021. Vol. 42, № 2. P. 115–130.

Информация об авторах

Медведев Максим Викторович, бакалавр, Пензенский государственный университет.

Антонов Илья Игоревич, бакалавр, Пензенский государственный университет.

Балашова Ирина Юрьевна, кандидат технических наук, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.4

JIT-КОМПИЛЯТОР NUMBA КАК ИНСТРУМЕНТ ДЛЯ РЕСУРСОЕМКИХ ВЫЧИСЛЕНИЙ

А. Э. Пилюгин[1], А. А. Харитонов[2], О. С. Дорофеева[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1] super.a-piliugin@yandex.ru
[2] a.kharitonovt.rex@gmail.com
[3] dos@pnzgu.ru

Аннотация. Демонстрируется использование языка Python с JIT (Just-In-Time) компилятором Numba для оптимизации вычислений на примере задачи расчета энергии фрагментов сигнала. Анализируются и сравниваются три подхода к решению этой задачи: последовательное вычисление на языке Python без дополнительных библиотек, последовательно вычисление с использованием JIT-компиляции и метод с применением CUDA-параллелизации. Каждый из подходов протестирован на сигналах разной длины с целью выявления наиболее эффективного решения для данной задачи.

Ключевые слова: оптимизация ресурсоемких вычислений, Python, Numba, JIT-компилятор, CUDA-параллелизация, энергия фрагментов сигнала

Для цитирования: Пилюгин А. Э., Харитонов А. А., Дорофеева О. С. JIT-компилятор Numba как инструмент для ресурсоемких вычислений // Вестник Пензенского государственного университета. 2024. № 4. С. 98–102.

Введение

Python – это высокоуровневый язык программирования с динамической типизацией [1]. JIT (Just-In-Time)-компиляция – это метод оптимизации, который позволяет преобразовывать код на Python в машинный код непосредственно перед выполнением. Для реализации использована библиотека Numba. Numba – это библиотека для Python, предназначенная для ускорения выполнения кода с помощью компиляции JIT [2]. Она использует компилятор LLVM и позволяет преобразовать Python-функции в машинный код, что значительно снижает задержки, связанные с интерпретацией. В данном исследовании применена JIT-компиляция с использованием декоратора @jit(nopython=True), что позволяет исключить взаимодействие с интерпретатором Python и обеспечить значительное ускорение выполнения кода.

Параллельные вычисления с использованием GPU позволяют обрабатывать большие объемы данных значительно быстрее, чем при работе на центральном процессоре (CPU). CUDA (Compute Unified Device Architecture) – это параллельная вычислительная платформа и программная модель, разработанная NVIDIA [3]. Она позволяет реализовать многопоточные вычисления на GPU, обеспечивая ускорение вычислений за счет параллельной обработки. Для реализации в данном исследовании использована библиотека numba.cuda, позволяющая выполнять вычисления непосредственно на GPU [4]. Функция для вычисления энергии фрагментов сигнала реализована с использованием декоратора @cuda.jit, который позволяет распределить вычисления между потоками GPU.

Основная цель исследования заключается в анализе использования JIT-компиляции Numba для ресурсоемких вычислений. В качестве примера взята задача вычисления энергии фрагментов сигнала.

Методы исследования

В рамках исследования были разработаны две программы, каждая из которых позволяет оценить влияние различных методов оптимизации на вычислительные задачи.

Задача состоит в вычислении энергии сигнала, представленного массивом, путем последовательного суммирования квадратов значений сигнала в каждом фрагменте (субцепочке) фиксированной длины. Эта энергия позволяет количественно оценивать мощность сигнала на выбранных интервалах, что важно для обработки и анализа сигналов. Основное ограничение данной задачи – значительные вычислительные затраты при увеличении длины сигнала, что требует эффективных методов оптимизации.

В первом примере программы осуществляется последовательное суммирование квадратов значений в каждом фрагменте сигнала сначала без дополнительной оптимизации, а потом с использованием JIT-компиляции (листинг 1).

Листинг 1. Последовательные вычисления

```
  import numpy as np
      import time
from numba import jit

# Оригинальная функция без JIT-компиляции
def calculate_energy(signal, sub_length):
        energy_sum = 0
        for i in range(len(signal) - sub_length + 1):
            sub_signal = signal[i:i + sub_length]
            energy = np.sum(np.square(sub_signal))
            energy_sum += energy
      return energy_sum

# JIT-оптимизированная функция
@jit(nopython=True)
def calculate_energy_jit(signal, sub_length):
      energy_sum = 0
      for i in range(len(signal) - sub_length + 1):
            sub_signal = signal[i:i + sub_length]
            energy = np.sum(np.square(sub_signal))
            energy_sum += energy
      return energy_sum

# Параметры тестирования
signal_length = 1000000
signal = np.random.choice([-1, 1], size=signal_length)
sub_lengths = [10, 50, 100, 200, 1000, 10000, 100000]  # Длины субцепочек

# Замер времени для обычной функции
print("Время выполнения без JIT:")
for sub_length in sub_lengths:
      start_time = time.time()
      energy_sum = calculate_energy(signal, sub_length)
```

```
        end_time = time.time()
            print(f"Субцепочка длиной {sub_length}: {end_time - start_time:.6f} секунд")
    # Замер времени для функции с JIT-компиляцией
    print("\nВремя выполнения с JIT:")
    for sub_length in sub_lengths:
        start_time = time.time()
        energy_sum_jit = calculate_energy_jit(signal, sub_length)
        end_time = time.time()
        print(f"Субцепочка длиной {sub_length}: {end_time - start_time:.6f} секунд")
```

Во втором примере программа совершает параллельное вычисление с использованием CUDA (листинг 2).

Листинг 2. Параллельное вычисление

```
import numpy as np
import time
from numba import cuda

# Функция для вычисления энергии с использованием CUDA
@cuda.jit
def calculate_energy_cuda(signal, sub_length, result):
    idx = cuda.grid(1)  # Получаем уникальный индекс каждого потока
    energy_sum = 0.0
    if idx < signal.size - sub_length + 1:  # Проверка границ
        for j in range(sub_length):        # Вычисление энергии в субцепочке
            energy_sum += signal[idx + j] ** 2
        result[idx] = energy_sum

# Параметры тестирования
signal_length = 1000000
# float32 для CUDA совместимости
signal = np.random.choice([-1, 1], size=signal_length).astype(np.float32)
sub_lengths = [10, 50, 100, 200, 1000, 10000, 100000]   # Длины субцепочек

# Выделяем память для результата
result = np.zeros(signal_length, dtype=np.float32)
d_signal = cuda.to_device(signal)
d_result = cuda.to_device(result)

# Замер времени для функции с CUDA
print("\nВремя выполнения с CUDA:")
for sub_length in sub_lengths:
    threads_per_block = 256
    blocks_per_grid = (signal_length + (threads_per_block - 1)) // threads_per_block
    start_time = time.time()
    calculate_energy_cuda[blocks_per_grid, threads_per_block](d_signal, sub_length, d_result)
    cuda.synchronize()  # Ожидание завершения всех потоков
    end_time = time.time()

    result = d_result.copy_to_host()  # Копируем результат обратно на CPU
      total_energy_sum = np.sum(result[:signal_length - sub_length + 1])  # Сумма энергии по всем фрагментам
    print(f"Субцепочка длиной {sub_length}: {end_time - start_time:.6f} секунд")
```

Тестирование программ осуществлялось на одних и тех же исходных данных. Для тестирования были выбраны сигналы длиной 1 000 000 и несколько длин фрагментов (субцепочек): 10, 50, 100, 200, 1000, 10 000, 100 000. Замеры времени выполнения осуществлялись с использованием библиотеки time.

Результаты исследования

Замеры времени выполнения программ с последовательным вычислением осуществлялись на персональном компьютере со следующими характеристиками:

- операционная система: Windows 11 Home;
- CPU: AMD Ryzen 5 5600 H 3.3 ГГц 64-bit;
- RAM: DDR4 1 × 16 Гб;
- ПО: PyCharm Community Edition 2024.1.2 с Python 3.10.

Измерение времени выполнения программы с CUDA-параллелизацией проводилось с помощью сервиса Google Colab. Это бесплатный онлайн-сервис от Google, который позволяет пользователям писать и выполнять код на Python в браузере [5]. Colab предоставляет доступ к вычислительным ресурсам, включая GPU.

Результаты измерений представлены на рис. 1.

Рис. 1. Результаты измерений

Очевидно, что программа на Python без оптимизации показала относительно низкую производительность. При увеличении длины сигнала и фрагментов времени выполнения увеличивается экспоненциально, что делает этот метод непригодным для задач, требующих высокой скорости обработки.

Напротив, JIT-компиляция (Numba) позволила достичь ускорения выполнения вычислений. При использовании субцепочки с длиной 100 000 время вычислений сократилось более чем в 2 раза по сравнению с программой без оптимизации.

Как видно по результатам измерений (см. рис. 1), CUDA-параллелизация обеспечила наибольшую производительность. Время выполнения вычислений сократилось в 36 раз по сравнению с последовательным вычислением без использования JIT-компиляции.

Заключение

Проведенное исследование показало, что использование JIT-компиляции и CUDA позволяет существенно повысить производительность при решении ресурсоемкой задачи расчета энергии фрагментов сигнала.

Список литературы

1. The Python Tutorial. URL: https://docs.python.org (дата обращения: 23.10.2024).
2. Open Source JIT compiler Numba Documentation. URL: https://numba.readthedocs.io/en (дата обращения: 25.10.2024).
3. CUDA Toolkit. URL: https://developer.nvidia.com (дата обращения: 30.10.2024).
4. Numba for CUDA GPUs. URL: https://docs.nvidia.com (дата обращения: 30.10.2024).
5. Google Colab. URL: https://colab.research.google.com (дата обращения: 25.10.2024).

Информация об авторах

Пилюгин Артём Эдуардович, студент, Пензенский государственный университет.

Харитонов Андрей Андреевич, студент, Пензенский государственный университет.

Дорофеева Ольга Станиславовна, кандидат технических наук, доцент, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.024

ОБЗОР МЕССЕНДЖЕРОВ ДЛЯ КОРПОРАТИВНОЙ СЕТИ

Д. В. Роганов [1], В. В. Эпп [2]

[1, 2] Пензенский государственный университет, Пенза, Россия

[1] danilok.03@mail.ru
[2] vitalinae@mail.ru

Аннотация. В целях оптимизации передачи информации внутри сети предприятия были проанализированы аналоги корпоративных мессенджеров. В ходе анализа требовалось выявить проблемы, а также то, что можно модифицировать в собственной разработке.

Ключевые слова: мессенджер, корпоративный мессенджер, предприятие

Для цитирования: Роганов Д. В., Эпп В. В. Обзор мессенджеров для корпоративной сети // Вестник Пензенского государственного университета. 2024. № 4. С. 103–105.

Корпоративные мессенджеры помогают всем членам команды координировать и организовывать деятельность в проектах, упрощают связь между отделами на предприятии или компании. На Западе большое количество таких мессенджеров, которые значительно повышают эффективность работы сотрудников, например, мессенджер Slack. Slack – один из самых известных корпоративных мессенджеров, в нем можно создавать каналы для отдельных проектов, обмениваться файлами, сообщениями, также есть возможность интеграции с различными сервисами. Есть также мессенджер Microsoft Teams. Это мессенджер от Microsoft, он сочетает в себе функции чата, есть возможность совместной работы в документах, а также возможность видеозвонков. Популярен Microsoft Teams среди компаний, которые используют остальную продукцию Microsoft. Zoom можно назвать корпоративным мессенджером, хоть он и известен платформой для видеозвонков, но в нем присутствует возможность совместной работы, функции чата.

Российские компании не отстают: в последнее время создают или улучшают отечественное программное обеспечение (ПО). У российских присутствуют такие же функции, как у западных. Большая часть российских мессенджеров внешне похожа на Telegram. В статье представлен анализ трех отечественных корпоративных мессенджеров.

Яндекс.Мессенджер. С 2022 г. перешел в категорию корпоративных мессенджеров. После этого в нем появились треды и реакции на сообщения. Как и в большинстве таких мессенджеров, интерфейс похож на Telegram. Если нужно разделить рабочие переговоры и личные сообщения, то он для этого подойдет. В основном используется на мобильных устройствах, но также есть ограниченная веб-версия. Подходит небольшим командам с запросом на обычные переписки с мобильных устройств. Платформы: веб-интерфейс, десктопные приложения (Windows, Mac, Linux), мобильные приложения. По подписке Яндекс 360 для бизнеса стоит от 249 руб./мес. [1] за одного пользователя. Из недостатков можно выделить: нужно обязательно покупать подписку на экосистему Яндекс, нельзя подключать интеграции, нет разделения на личные и групповые чаты, нет видеозвонков.

eXpress. На первый план выносится безопасность. Можно устанавливать на свои сервера, весь трафик проходит по двуслойному шифрованию, есть криптоконтейнеры. В eXpress есть свой каталог ботов и различных приложений, но также есть возможность создания своих. Подходит большим командам или госкомпаниям, которым в приоритет ставится защита. Платформы: веб-интерфейс, десктопные приложения(Windows, Mac, Linux), мобильные приложения, можно установить на облако или сервер. Для одиночных пользователей он бесплатный, в рабочем пространстве 3000 руб./год [2] за сотрудника, также имеется индивидуальный расчет стоимости. Преимущества: повышенная систем безопасности, есть гостевой доступ, возможность общаться с внешними пользователями, как в обычном мессенджере. Из недостатков можно выделить то, что нет возможности управлять группами пользователей, сложное внедрение и нет разделения на личные и групповые чаты.

VK Teams. Корпоративный мессенджер от ВКонтакте. Базовый функционал реализован хорошо, а интерфейс похож на ВК, что упрощает знакомство с этим мессенджером. Отличий от бесплатной и платной версии нет. Но в платной версии есть внутренний постановщик задач, календарь и треды. Подходит как небольшим компаниям, так и крупным, которые хотят объединить общение отделов в одном сервисе. Платформы: веб-интерфейс, десктопные приложения (Windows, Mac, Linux), мобильные приложения, можно установить на облако или сервер. Базовый функционал предоставляется бесплатно (до пяти сотрудников), до 500 сотрудников – 199 руб./мес. за человека, от 500 сотрудников – цена договорная и рассчитывается индивидуально [3]. Из преимуществ: интеграция с API, 100 ГБ места на пользователя в облачном хранилище, видеоконференции. К недостаткам можно отнести: сложную регистрацию, не самый удобный интерфейс, большая часть полезных функций предоставляется только в платной версии, бесплатная версия с рекламой и только до пяти сотрудников.

В ходе анализа можно выявить, что большая часть российских мессенджеров предоставляет бесплатный функционал, а также кроссплатформенность. Есть возможности интеграции с другими корпоративными системами, которые необходимы для лучшей эффективности работы команды. Но в современном мире большее внимание надо уделять безопасности данных, которые передаются внутри корпоративных мессенджеров, так как большая часть компаний работает в условиях повышенных требованиях к конфиденциальности. Результаты анализа представлены в табл. 1.

Таблица 1

Сравнение мессенджеров для корпоративной сети

Критерии	Яндекс.Мессенджер	eXpress	VK Teams
Безопасность	Базовая	Трафик проходит по двуслойному шифрованию, есть криптоконтейнеры	Антивирус Касперского, защита от спама и фишинга, двухфакторная аутентификация
Платформы	Веб-интерфейс, десктопные приложения (Windows, Mac, Linux), мобильные приложения	Веб-интерфейс, десктопные приложения (Windows, Mac, Linux), мобильные приложения, можно установить на облако или сервер	Веб-интерфейс, десктопные приложения (Windows, Mac, Linux), мобильные приложения, можно установить на облако или сервер
Стоимость	От 249 руб./мес.	3000 руб./год	Бесплатно до пяти сотрудников, до 500 сотрудников – 199 руб./мес. за человека, от 500 сотрудников – цена договорная и рассчитывается индивидуально
Интеграции с другими средствами	Нет	Есть	Есть

Из данной таблицы можно сделать вывод, что не все продукты предоставляют необходимую предприятию или компании защиту данных, стоимость продукта пропорциональна качеству, у всех есть кроссплатформенность, что делает их универсальными, у большинства есть интеграции с другими программными средствами для улучшения эффективности выполнения задач.

Список литературы

1. Статья «Лучшие корпоративные мессенджеры 2024 года». URL: https://vc.ru (дата обращения: 29.10.2024).
2. Система коммуникаций «Express». URL: https://express.ms/prices (дата обращения: 29.10.2024).
3. VK WorkSpace. URL: https://biz.mail.ru (дата обращения: 29.10.2024).

Информация об авторах

Роганов Данила Вячеславович, студент, Пензенский государственный университет.

Эпп Виталина Викторовна, кандидат технических наук, доцент, доцент кафедры «Системы автоматизированного проектирования», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.4'423

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ИМПЛЕМЕНТАЦИЙ ИНТЕРПРЕТАТОРОВ НА ПРИМЕРЕ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ PYTHON И PHP

Н. Д. Самсонов[1], А. В. Барабанов[2], О. С. Дорофеева[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1] skiipering@gmail.com
[2] bandreyv04@yandex.ru
[3] dos@pnzgu.ru

Аннотация. Рассматриваются основные этапы работы интерпретатора языков программирования Python и PHP, производится сравнительный анализ принципов их работы для выявления схожих или различных подходов к реализации, наличия ее особенностей с учетом основной сферы применения для подтверждения или опровержения схожести характеристик языков в зависимости от методов их работы. Сделан вывод о схожих принципах работы интерпретаторов языков и разных временных рамках решения одинаковых задач, отдельное внимание уделено вопросу удобства отладки на этапах интерпретирования.

Ключевые слова: интерпретатор, лексический анализатор, синтаксический анализатор, байт-код, виртуальная машина, PHP, Python

Для цитирования: Самсонов Н. Д., Барабанов А. В., Дорофеева О. С. Сравнительный анализ имплементаций интерпретаторов на примере языков программирования Python и PHP // Вестник Пензенского государственного университета. 2024. № 4. С. 106–110.

Интерпретатор – разновидность транслятора, который переводит исходный код в машинные инструкции во время работы программы. Интерпретаторы обеспечивают построчную обработку кода, обычно включающую в себя несколько этапов: лексический анализ, или токенизация, синтаксический разбор, семантический анализ, преобразование исходного кода в промежуточный, выполнение кода на виртуальной машине. На этапе токенизации обрабатываемая строка разбивается на более мелкие единицы – лексемы, которые представляют собой наиболее малые значимые элементы: литералы, операторы, ключевые слова [1]. Далее на этапе синтаксического разбора интерпретатор определяет, соответствуют ли полученные токены синтаксическим правилам языка, посредством анализа структуры кода, включающего лексемы, для построения дерева разбора или абстрактного синтаксического дерева (AST) [2]. Абстрактное синтаксическое дерево представляет собой древовидную структуру из узлов, где каждый из них является элементом языка: переменной, оператором или ключевым словом. Главным отличием дерева разбора от абстрактного синтаксического дерева является то, что у первого отсутствует абстракция узлов дерева от деталей конкретного языка, что обеспечивает подробное отображение структуры для более продуктивного отлаживания. На третьем этапе происходит семантический анализ, т.е. проверка на корректность и совместимость, например, проверка на то, что используемая переменная определена, или совместимость типов для какой-либо операции. Далее из удовлетворяющей условиям языка древовидной структуры генерируется промежуточный код. Это низкоуровневые, не зависимые

от платформы инструкции, предназначенные для корректного выполнения на виртуальной машине. В данном случае независимость от реализации какой-либо платформы обеспечивает языку кроссплатформенную совместимость и портативность. На заключительном этапе промежуточный код, сгенерированный для обработанной строки, передается в виртуальную машину, управляющую памятью и выполнением инструкций.

При выборе интерпретируемого языка для решения определенных задач играет особую роль способ реализации самого интерпретатора. Среди интерпретируемых языков широкую известность получили Python и PHP как идеальные языки для сфер своего применения. Python используется для вычислений в задачах различных научных областей, анализа данных и машинного обучения, где его интерпретатор эффективно работает с логикой общего назначения и сложными алгоритмами. PHP предназначен для серверной веб-разработки, его использует большинство сайтов. Таким образом, выбор между данными языками предполагает понимание того, как их интерпретаторы оптимизированы для решения конкретных задач, а также для продвинутой отладки.

В обоих языках лексический анализатор разбивает исходный код на лексемы, этот процесс является основополагающим для обоих языков как начальный этап для подготовки кода к анализу. Далее синтаксический анализатор строит абстрактное синтаксическое дерево, обрабатывая последовательность полученных токенов. Стоит отметить, что и Python, и PHP используют именно абстрактное дерево, что делает поведение обоих интерпретаторов на начальных шагах крайне схожим. Далее в табл. 1 приводится пример работы лексического и синтаксического анализаторов в языках Python и PHP.

Таблица 1

Интерпретация исходного кода лексическими и синтаксическими анализаторами Python и PHP

Название языка	Исходный код	Лексический разбор	Синтаксический разбор
Python	x = 5	TokenInfo(type=1 (NAME), string='x', start=(1, 0), end=(1, 1), line='x = 5') TokenInfo(type=54 (OP), string='=', start=(1, 2), end=(1, 3), line='x = 5') TokenInfo(type=2 (NUMBER), string='5', start=(1, 4), end=(1, 5), line='x = 5') TokenInfo(type=4 (NEWLINE), string='', start=(1, 5), end=(1, 6), line='') TokenInfo(type=0 (ENDMARKER), string='', start=(2, 0), end=(2, 0), line='')	Module(body=[Assign(targets=[Name(id='x', ctx=Store())], value=Constant (value=5), type_comment=None)], type_ignores=[])

PHP	<?php
	$x = 5;
	?>
		T_OPEN_TAG: <?php
		T_WHITESPACE:
		T_VARIABLE: $x
		T_WHITESPACE: Symbol: =
		T_WHITESPACE:
		T_LNUMBER: 5 Symbol: ;
		T_CLOSE_TAG: ?>
			array(1) {
			[0] => ,Stmt_Expression(
			expr: Expr_Assign(
			var: Expr_Variable(
			name: x
			)
			expr: Scalar_LNumber(
			value: 5
			)
			)
			)
			}

Однако после проведения семантического анализа интерпретаторы начинают работать с имеющимися данными по-разному. Python преобразует AST в байт-код, являющийся набором инструкций для Python Virtual Machine. Для избежания повторных преобразований одного и того же кода Python предусматривает кеширование байт-кода в файлы с расширением «.pyc» в директории __pycache__. Если при запуске скрипта не будет выявлено изменений, то стадия генерации байт-кода будет пропущена. Ниже, на рис. 1, в левой части представлены исходный код программы и вызов генератора байт-кода для описанной функции в правой части.

Рис. 1. Пример генерации байт-кода для функции в языке Python

После успешной генерации байт-кода он обрабатывается виртуальной машиной. Виртуальная машина Python использует стек, в который инструкции добавляют и удаляют значения, чтобы выполнять операции. Машина обрабатывает байт-код построчно, управляя потоком, памятью, и манипулирует данными. Для кода, изображенного на рис. 1, операции «LOAD_CONST» и «LOAD_FAST» используются для загрузки значений в стек, а «STORE_FAST» указывает на то, что в переменной будет храниться значение [3].

В интерпретаторе PHP перевод осуществляется в промежуточную форму, называемую «opcodes», для выполнения с помощью Zend Engine. Они также являются не зависимыми от платформы и являют собой базовые операции присваивания, арифметики и вызовов. Далее на рис. 2 в левой части приведен исходный код на PHP, в правой части указаны «opcodes».

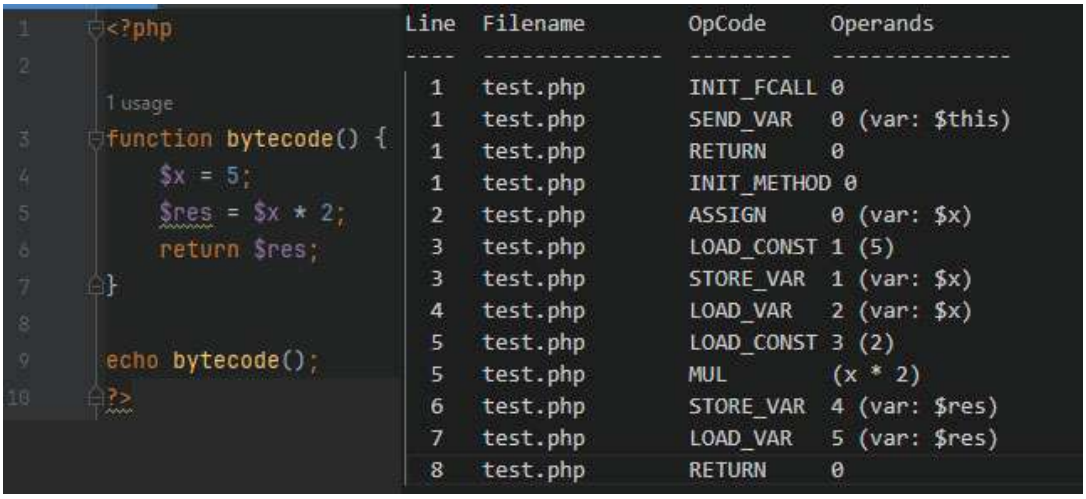


Рис. 2. Пример генерации «opcodes» для функции в языке PHP

Zend Engine также использует стек для хранения промежуточных значений и управления потоком. На рис. 2 можно увидеть коды «STORE_VAR» и «LOAD_VAR», действующие аналогично кодам «STORE_FAST» и «LOAD_FAST» в байт-коде Python. Zend Engine дополнительно предполагает автоматическое сворачивание констант, т.е. если в выражении фигурирует константа, то значение вычисляется сразу и оптимизируется таким образом, чтобы выражение содержало наименьшее количество операций.

Исходя из вышеописанной информации, можно сделать вывод о схожей реализации интерпретаторов PHP и Python на определенном уровне абстракции, несмотря на разные сферы применения. Однако, исходя из этих данных, нельзя заранее думать, что скорость их работы для решения тривиальных задач будет одинакова. Для показательного сравнения скорости работы интерпретаторов были проведены тесты заполнения пяти коллекций разных величин случайными элементами. Результаты этих тестов приведены в табл. 2.

Таблица 2

Сравнение скорости заполнения коллекции случайными элементами

Количество элементов	Время работы Python (с)	Время работы PHP (с)
1000	0,001922	0,00004
10 000	0,005596	0,000370
100 000	0,030364	0,003554
1 000 000	0,281712	0,034734
100 000 000	27,650419	3,483425

Из результатов теста видно, что Python значительно уступает PHP в скорости, вследствие чего можно говорить о том, что схожесть реализаций интерпретаторов не обязательно означает, что скорость выполнения программ, описанных на этих языках, будет одинаковой.

Таким образом, несмотря на разные сферы применений языков, PHP и Python имеют схожую структуру интерпретаторов, где главным отличием является разный подход к реализации кода промежуточного уровня. Стоит учесть, что получаемые интерпретатором Python результаты на разных этапах обработки кода лучше читаются и удобнее структурированы для восприятия человеком, что способствует более продуктивной отладке кода на низком уровне, если в этом есть необходимость, при этом все отладочные функции и профилирование доступны в базовой версии языка, в то время как для отладки кода PHP на том же уровне необходимо самостоятельно подключать и конфигурировать необходимый инструментарий.

Принимая вышеизложенную информацию во внимание, можно констатировать, что структура интерпретаторов, работающих с байт-кодом, типична и имеет различия только в нюансах реализации, таких как виды представления синтаксического разбора, реализация байт-кода и вопросы оптимизации виртуальной машины для исполняемого кода. В то же время, зная эту информацию, нельзя поспешно делать вывод о том, что скорость выполнения программы на разных интерпретируемых языках схожей реализации будет варьироваться минимально.

Список литературы

1. Ахо А. В., Лам М. С., Сети Рави, Ульман Дж. Д. Компиляторы: принципы, технологии и инструментарий. М. : Диалектика-Вильямс, 2017. 1184 с.

2. Ахо А. В., Ульман Дж. Д. Теория синтаксического анализа, перевода и компиляции / под ред. В. М. Курочкина. М. : Мир, 1978. 612 с.

3. Obi Ike-Nwosu, Inside The Python Virtual Machine // Leanpub. URL: https://leanpub.com

Информация об авторах

Самсонов Никита Денисович, бакалавр, Пензенский государственный университет.

Барабанов Андрей Валерьевич, бакалавр, Пензенский государственный университет.

Дорофеева Ольга Станиславовна, кандидат технических наук, доцент, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.05

ОБЗОР И СРАВНЕНИЕ АЛГОРИТМОВ БИНАРНОГО ПОИСКА

С. В. Самуйлов[1], С. В. Самуйлова[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1]sws_p@mail.ru
[2]s_v_samuylova@mail.ru

Аннотация. Рассмотрены наиболее известные алгоритмы бинарного поиска: классический бинарный поиск, оптимальный бинарный поиск, интерполяционный поиск и «последовательный» бинарный поиск. Выполнен сравнительный анализ этих алгоритмов.

Ключевые слова: поиск, классический бинарный поиск, оптимальный бинарный поиск, интерполяционный бинарный поиск, «последовательный» бинарный поиск, сравнительный анализ алгоритмов, эффективность алгоритмов обработки данных

Для цитирования: Самуйлов С. В., Самуйлова С. В. Обзор и сравнение алгоритмов бинарного поиска // Вестник Пензенского государственного университета. 2024. № 4. С. 111–113.

В настоящее время, когда объем окружающей нас информации удваивается в течение одной жизни человека, актуальными являются знание эффективных алгоритмов поиска и умение целенаправленно применять лучший по характеристикам алгоритм поиска в зависимости от условий поиска.

Когда речь идет о поиске, то следует подчеркнуть, что эффективному поиску всегда должна предшествовать некоторая организация данных. Если данные не организованы, т.е. порядок расположения элементов последовательности неизвестен, то единственный вариант поиска – последовательный перебор элементов последовательности, другими словами, последовательный поиск. Алгоритмы данного типа очень неэффективны. В лучшем случае при удачном поиске число сравнений в среднем равно $n/2$, где n – размерность последовательности [1].

Как уже было отмечено выше, единственный способ улучшения данного показателя – предварительная организация данных. Существуют несколько способов организации данных.

Очень неплохие результаты дает организация данных с помощью деревьев. Это могут быть и бинарные деревья поиска, и сбалансированные деревья, и B-деревья, и т.п. Как правило, алгоритмы поиска при такой организации имеют сложность O(log (n)).

Другой подход к организации данных – построение хеш-таблиц, предполагающее использование методов преобразования ключа в адрес (использование хеш-функций) как на этапе организации данных, так и на этапе их поиска. Эффективность таких алгоритмов во многом зависит как от выбранной хеш-функции, так и от метода разрешения коллизий. В частности, при использовании метода цепочек такой подход может давать очень хорошие результаты при поиске.

В данной статье рассматривается один из самых часто используемых способов организации данных – их упорядочение. Если исходная последовательность отсортирована, то для поиска может быть использован один из алгоритмов бинарного поиска.

Самый известный – классический алгоритм бинарного поиска – основан на последовательном сравнении ключа поиска со средним элементом оставшейся упорядоченной последовательности. Если значения совпадают, поиск успешно заканчивается. Несовпадение значений позволяет сократить количество анализируемых элементов сразу вдвое. Сложность такого алгоритма определяется как O(log (*n*)) [2].

Другой алгоритм бинарного поиска, имеющий название оптимального бинарного поиска, позволяет сделать такой поиск чуть быстрее за счет сокращения операций, выполняемых в цикле поиска. Однако и этот алгоритм имеет ту же сложность O(log (*n*)) [3].

Другим значительным улучшением алгоритма классического бинарного поиска является предположение, что если значения в интервале значений располагаются достаточно равномерно, то местоположение ключа поиска может быть определено более точно на основании значения самого ключа, а также значений граничных элементов и количества элементов последовательности. Такой подход, называемый интерполяционным поиском, значительно сокращает число сравнений и имеет сложность O(log (log (*n*))) [4].

Описанные выше алгоритмы бинарного поиска являются хорошо известными и часто используемыми. Однако существует еще один алгоритм бинарного поиска, на наш взгляд, незаслуженно мало применяемый на практике. Алгоритм заключается в следующем. Исходная отсортированная последовательность просматривается слева направо с помощью фиксированных перемещений – прыжков. Сначала длина прыжка равна *n*/2, где *n* – размерность последовательности. Затем прыжок уменьшается вдвое до тех пор, пока не станет равным нулю [5].

При этом прыжок возможен только при выполнении двух условий: прыжок не выполняется за пределы последовательности, и прыжок не выполняется на элемент, превышающий по значению искомый ключ. Пока прыжок возможен, прыгаем. Если невозможен, уменьшаем размер прыжка и снова прыгаем. Как только размер прыжка станет равен нулю, проверяем наличие ключа поиска в текущей позиции.

Сам алгоритм имеет небольшой и интуитивно понятный код (рис. 1) и по сложности написания программы значительно более простой, чем описанные выше алгоритмы бинарного поиска. Назовем этот алгоритм «последовательным» бинарным поиском. Эта модификация алгоритма бинарного поиска также имеет сложность O(log (*n*)).

```
int k = 0;          // исходная позиция
int b = n / 2;      // длина прыжка
while(b>0)
        {
        while ((k + b < n) && (m[k + b] <= x)) k += b;
        b = b / 2;
        }
if (m[k] == x) {
        cout << "Нашли" << endl;}
else    {cout << "Не нашли" << endl;}
```

Рис.1. «Последовательный» бинарный поиск

Так как формулы оценки сложности алгоритмов достаточно грубы, для определения практической эффективности рассмотренных выше алгоритмов бинарного поиска была написана программа [6], в которой было выполнено их сравнение (рис. 2).

Анализ работы алгоритмов бинарного поиска показывает, что безусловным лидером среди алгоритмов является интерполяционный поиск, но и «последовательный» бинарный поиск в среднем дает результаты в два раза и более эффективнее классического и оптимального бинарных поисков.

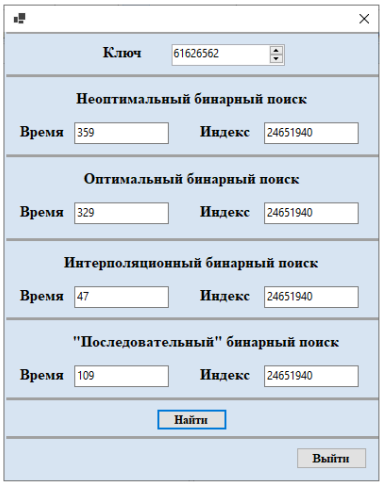


Рис. 2. Сравнение алгоритмов бинарного поиска

Список литературы

1. Самуйлов С. В., Самуйлова С. В. Структуры данных. Алгоритмы поиска и сортировки : учеб. пособие. М., 2024. doi: 10.23682/139334

2. Бинарный поиск. URL: https://blog.skillfactory.ru/glossary/binarnyj-poisk/ (дата обращения: 22.10.2024).

3. Информатика. Бинарный поиск. URL: https://bstudy.net/794516/informatika/binarnyy_poisk (дата обращения: 22.10.2024).

4. Интерполяционный поиск элемента в массиве. URL: http://algolist.ru/search/int_search.php (дата обращения: 22.10.2024).

5. Лааксонен А. Олимпиадное программирование / пер. с англ. А. А. Слинкин. М. : ДМК Пресс, 2018. 300 с.

6. Свидетельство о регистрации программы для ЭВМ RU 2020611961. Обучающая программа по алгоритмам поиска / Самуйлов С. В., Самуйлова С. В.; заявка № 2020610735 от 30.01.2020; опубл. 12.02.2020.

Информация об авторах

Самуйлов Сергей Владимирович, кандидат технических наук, доцент, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Самуйлова Светлана Валентиновна, кандидат технических наук, доцент, доцент кафедры «Высшая и прикладная математика», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.42

НАПИСАНИЕ REST-API СЕРВИСА РЕКОМЕНДАЦИЙ НА SWI-PROLOG

Н. С. Сахно[1], А. А. Новосельцев[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1] niksahn@gmail.com
[2] sasha_novosel@mail.ru

Аннотация. Рассматривается создание REST-API сервера на языке логического программирования Prolog в его реализации SWI-Prolog. Даны определения таким основным инструментам, использовавшимся при разработке сервера, как REST-API, Prolog, SWI-Prolog. Кроме того, приводятся библиотеки, используемые непосредственно при работе, а также рекомендации по использованию языка Prolog при разработке сервера и основные минусы такой работы.

Ключевые слова: Prolog, SWI-Prolog, REST, веб-разработка, сервис рекомендаций

Для цитирования: Сахно Н. С., Новосельцев А. А. Написание REST-API сервиса рекомендаций на SWI-Prolog // Вестник Пензенского государственного университета. 2024. № 4. С.114–116.

Процесс создания современного приложения не обходится без реализации его серверной части, в которой и заложены логика и основной функционал. Сегодня, выбирая средства для написания сервера, разработчики отдают предпочтение языкам объектно-ориентированного программирования (ООП), некоторые из которых спустя десятилетия не теряют своей популярности и со временем приобретают статус единственного верного решения [1]. Однако для данных целей можно использовать языки иных парадигм программирования. Мы будем рассматривать написание сервера рекомендаций на языке логического программирования SWI-Prolog. Язык Prolog часто используется для обработки символьной информации и для различных сложных логических задач. Например, его можно легко применить к задаче построения рекомендательных систем, особенно в случаях, когда требуется обработка сложных логических условий или правил. Под рекомендательной системой будем подразумевать программу, которая пытается предсказать, какие объекты будут интересны пользователю, имея определенную информацию о его профиле.

REST-API – два часто встречающихся акронима (сокращение с англ.), которые расшифровываются как Representational State Transfer и Application programming interface. Итак, REST – это архитектурный стиль взаимодействия элементов распределенной системы. Так называется стиль взаимодействия между отдельными частями системы, которые необязательно должны быть расположены в одном месте. Другими словами, это специальный стиль, обладающий набором ограничений, которые должны учитываться на этапах проектирования системы [2].

Пролог – это язык программирования, предназначенный для обработки символьной нечисловой информации. Особенно хорошо он приспособлен для решения задач, в которых фигурируют объекты и отношения между ними [3].

SWI-Prolog – свободная реализация языка программирования Пролог. Несмотря на то, что SWI-Prolog может показаться нестандартным выбором для создания серверной части приложения,

его встроенные библиотеки и инструменты позволяют легко разрабатывать веб-приложения, включая REST-API.

SWI-Prolog поставляется с мощной библиотекой для работы с HTTP-запросами и построения веб-сервисов. Эта библиотека включает поддержку как HTTP-клиента, так и HTTP-сервера, что делает ее отличным инструментом для создания REST-API. Для создания REST-API можно использовать следующие модули: library(http/thread_httpd) – для запуска HTTP-сервера, library(http/http_dispatch) – для маршрутизации запросов, library(http/http_json) – для работы с JSON-данными

Преимущество Prolog заключается в его декларативной природе и встроенной логике поиска решений, что делает его хорошим выбором для задач, связанных с рекомендациями. Достаточно объявить динамические факты и на основе правил строить рекомендации. Также можно организовать взаимодействие между этим сервисом и клиентом, которым может быть другой сервис, используя REST-API протокол.

Допустим, правила, по которым пользователю рекомендуется конкретный фильм, представляются следующим образом:

% Объявляем динамические факты, на основании которых будем строить рекомендации

Листинг 1. Правило для рекомендации фильма

```
:- dynamic watched/2. % Пользователю понравился фильм
:- dynamic genre/2. % Жанр фильма
recommend(User, Movie) :-
watched(User, LikedMovie),
genre(LikedMovie, Genre),
genre(Movie, Genre),
\+ watched(User, Movie).
```

Мы можем организовать взаимодействие между этим сервисом и клиентом, которым может быть другой сервис, используя REST-API протокол.

Клиент будет передавать нам в телеPOST запроса данные о понравившихся пользователям фильмах в виде json-формата:

```
{
 "users": [ {"user": "Пользователь", "movie": "Понравившийся фильм"}],
 "films": [ {"movie": "Понравившийся фильм", "genre": "Sci-Fi"} ]
}
```

Ответ от сервера будет json-формата:

[“user”: ”Пользователь”, recommendation:”Рекомендуемый фильм”]

Для этого в коде программы на Прологе объявим обработчик post-запроса по пути /recommend, им будет процедура recommend_handler.

Сам обработчик запроса будет добавлять полученные факты в базу знаний и на основе фактов из этой базы строить рекомендацию (листинг 2).

Листинг 2. Основной код программы

```
:- http_handler(root(recommend), recommend_handler, [method(post)]).
recommend_handler(Request) :-
 http_read_json_dict(Request, Dict),
  add_facts(Dict),
 findall(json([user=User, recommendation=Recommendation]), recommend(User, Recommendation), Recommendations),
  reply_json(Recommendations).
% Добавить факты в базу знаний на основе словаря
```

```
add_facts(Dict) :-
   forall(member(UserDict, Dict.users),
        (atom_string(User, UserDict.user),
         atom_string(Movie, UserDict.movie),
         ( \+ watched(User, Movie) -> assertz(watched(User, Movie)) ; true ))),
   forall(member(FilmDict, Dict.films),
        (atom_string(Movie, FilmDict.movie),
         atom_string(Genre, FilmDict.genre),
         ( \+ genre(Movie, Genre) -> assertz(genre(Movie, Genre)) ; true ))).
```

Для развертывания приложения в системе Docker также существует готовый образ, поэтому создание и запуск контейнера не составят труда.

Таким образом, мы можем получить упрощенный сервер для рекомендаций на языке SWI-Prolog. На этом возможности Пролог в веб-разработке не ограничиваются. Пролог имеет некоторые библиотеки для работы с сетью, например, библиотеки для работы с системой Redis, различные библиотеки для работы с базами данных. Однако, несмотря на все это, лучше всего использовать Пролог в качестве вспомогательного языка при разработке веб-системы, а в качестве основного языка использовать один из императивных, более подходящих для этого, так как данный язык имеет ряд серьезнейших недостатков, выявленных при разработке данного сервиса, среди которых можно выделить следующие. В отличие от других популярных языков, у Пролога нет широко поддерживаемых веб-фреймворков, хотя есть библиотеки, такие как SWI-Prolog HTTP library, они далеко не так развиты. Сама парадигма логического программирования не всегда удобна и не широко распространена среди разработчиков. Также в Прологе обработка большого количества запросов и работа с асинхронными операциями менее эффективны, чем в специализированных для веб-разработки языках.

Создание REST-API сервиса рекомендаций на SWI-Prolog – это интересная и полезная задача, которая демонстрирует гибкость этой логической системы программирования. Используя встроенные библиотеки для работы с HTTP и JSON, можно легко разрабатывать веб-сервисы и API. SWI-Prolog может быть использован в широком спектре задач, а создание REST-API на нем может пригодиться для интеграции этого языка в современные системы. Несмотря на это, данный язык довольно сложно использовать в качестве основного при разработке крупных онлайн-систем, в связи с его минусами для веб-разработки.

Список литературы

1. TIOBE Index for October 2024. URL: https://www.tiobe.com

2. Иванов А. О REST API простыми словами. URL: https://smoff.ru

3. Братко И. Программирование на языке Пролог для искусственного интеллекта. М. : Мир, 1990. С. 10.

Информация об авторах

Сахно Никита Сергеевич, бакалавр, Пензенский государственный университет.

Новосельцев Александр Андреевич, бакалавр, Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.4'413

СРАВНЕНИЕ РЕАЛИЗАЦИЙ СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА ЯЗЫКА ПРОГРАММИРОВАНИЯ НА C# И JAVASCRIPT

И. А. Смирнов[1], М. А. Пичушкина[2], О. С. Дорофеева[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1] IlySmir3@yandex.ru
[2] rita.pichushkina.04@mail.ru
[3] dos@pnzgu.ru

Аннотация. Проводится сравнительный анализ разработки синтаксических анализаторов для *LL*(1) грамматики на языках C# и JavaScript. Подробно рассматриваются принципы и правила *LL*(1) грамматики, а также этапы и составляющие для построения синтаксического анализатора (включая использование *LL*(1) и *LR*(1) грамматик, лексический анализатор, работу автомата с магазинной памятью и принцип построения дерева синтаксического разбора). Рассматриваются проблемы и преимущества реализации данных компонентов на двух языках, придерживающихся разных парадигм программирования – объектно-ориентированной в C# и прототипно-ориентированной в JavaScript – и влияние данных парадигм на процесс разработки. Рассматривается влияние производительности и скорости разработки на выбор языка. На основе проанализированной информации делается вывод о том, какой из языков будет иметь преимущество в выборе при разработке синтаксического анализатора.

Ключевые слова: синтаксический анализатор, лексический анализатор, парсер, грамматика языка, LL(1) грамматика, детерминированный магазинный автомат, C#, JavaScript, Node.js, .Net

Для цитирования: Смирнов И. А., Пичушкина М. А., Дорофеева О. С. Сравнение реализаций синтаксического анализатора языка программирования на C# и JavaScript // Вестник Пензенского государственного университета. 2024. № 4. С. 117–121.

При разработке своего языка программирования одним из важных этапов является создание синтаксического анализатора (парсера). Неизбежно будут возникать вопросы о том, как же лучше всего его реализовать, ведь в настоящее время существует множество языков, предоставляющих разные инструменты и разный подход к разработке. Произведем сравнение двух подходов к реализации синтаксического анализатора – с помощью языков C# и JavaScript.

Для проведения сравнительного анализа реализации детерминированного нисходящего *LL*(1) магазинного автомата, представляющего собой синтаксический анализатор, на двух разных языках программирования необходимо определить основные понятия.

Грамматикой называется набор правил, описывающих порядок формирования слов из символов алфавита языка, соответствующих синтаксису языка. При этом грамматика не описывает значение слов или то, в каком контексте слова и предложения языка можно использовать.

LL(1) грамматика является подмножеством контекстно-свободных (КС) грамматик, главной особенностью которых является наличие в левых частях всех правил только одиночных нетерминалов. Название семейства *LL*(*k*) заключает в себе основные правила работы данной грамматики.

В данном случае первая буква *L*(*Left-to-Right* – слева направо) обозначает, что разбор строки будет идти слева направо. Вторая буква *L*(*Leftmost derivation* – самая левая последователь-

ность) показывает, что разбор данной грамматики проходит леворекурсивно, т.е. нетерминалы в правой части правил заменяются, начиная с самых левых. Леворекурсивность *LL*(1) грамматики является главной отличительной чертой по сравнению с другим подмножеством КС-грамматик. Грамматика *LR*(*k*) придерживается принципа *Rightmost derivation* (самая правая последовательность), обеспечивающего замену нетерминалов, начиная с самых правых [1]. Суффикс (*k*) показывает, какое количество символов вперед просматривает парсер перед принятием решения о выборе правила, где *k* – это количество таких символов.

В нашем случае используется грамматика типа *LL*(1), и она требует отсутствие левой рекурсии в правилах, так как выбор правила осуществляется на основе только одного символа, следовательно, при наличии левой рекурсии парсер не сможет определить применяемое правило. За счет этого обеспечиваются детерминированность в грамматике и создание детерминированного магазинного автомата [1].

Таким образом, парсеры на основе *LL*(1) грамматики будут простыми в реализации, так как данная грамматика исключает возможность левой рекурсии, а также подразумевает детерминированность и однозначность. В то же время это является существенным ограничением, так как грамматика такого вида не способна описать все контекстно-свободные языки, требующие большего количество просматриваемых парсером символов и более сложных механизмов разбора, например, левой рекурсии.

Конечным автоматом является абстрактная модель, которая содержит конечное количество состояний и используется для представления и управления потоком выполнения каких-либо команд. Магазинный автомат представляет из себя конечный автомат, расширенный стеком. Детерминированность магазинного автомата подразумевает, что в любом состоянии системы при наличии входного символа автомат имеет только один способ перехода в следующее состояние [2]. Использование *LL*(1) грамматики в основе построения магазинного автомата автоматически делает его детерминированным за счет своего свойства.

Перед этапом синтаксического анализа выполняет свою работу лексический анализатор. Лексический анализатор является первым этапом в разборе входной строки. Он определяет ошибки в ключевых словах, сторонние символы и все прочее, что связано с лексикой языка. После отработки этого этапа на выходе мы получаем список токенов, который затем поступит на вход синтаксическому анализатору [3].

Синтаксический анализатор является компонентом компилятора или интерпретатора, который проверяет данные, поступающие на вход программы, на соответствие грамматике языка и строит синтаксическое дерево. Одной из возможных реализаций синтаксического анализатора может являться нисходящий магазинный автомат. В таком случае тип синтаксического анализатора будет также нисходящим, что подразумевает построение синтаксического дерева сверху вниз, начиная с начального символа. Данный тип построения обеспечивается выбранной грамматикой. В случае использования грамматики *LL*(1) дерево строится вышеуказанным способом, так как разбор грамматики происходит леворекурсивно [4]. Синтаксические анализаторы, основанные на *LL*(1) грамматике, также могут называться *LL*(1) парсерами.

Напротив, для грамматики типа *LR*(*k*) построение дерева проходило бы снизу вверх [4]. Такая реализация синтаксического анализатора является более мощной и способной поддерживать большее количество языков, например, за счет поддержки левой рекурсии в грамматике, однако в то же время она более сложна в реализации.

Наличие стека в магазинном автомате позволяет хранить символы грамматики для последующего сопоставления с входными символами. Так как нисходящий разбор начинается с начального символа, то он помещается изначально в стек, после чего начинается процесс замены нетерминалов на их правые части в соответствии с грамматикой. Важно отметить, что добавление в стек происходит в обратном порядке для обеспечения принципа *LIFO* (последним при-

шел, первым ушел). Замена нетерминалов происходит до тех пор, пока верхний элемент в стеке не окажется терминалом, соответствующим токену входной строки, полученному на этапе лексического анализа. После этого терминал становится листом дерева разбора, так как его значение окончательно было интерпретировано в контексте программы. Таким образом, по завершении работы на выходе парсера мы получаем синтаксическое дерево.

Для сравнения реализации синтаксического анализатора были выбраны такие языки, как C# и JavaScript. Оба языка популярны и используются для различных проектов. Однако их особенности применения и парадигмы разнятся, что делает сравнение реализованных на них синтаксических анализаторов интересным.

JavaScript – это язык, используемый преимущественно для веб-разработки. Он используется повсеместно в браузерах для программирования пользовательской части сайтов, однако его можно использовать и на серверной части. Этот язык является прототипно-ориентированным. Прототипно-ориентированное программирование является ответвлением объектно-ориентированной парадигмы. Отличие между ними заключается в том, что в прототипно-ориентированном программировании объекты создаются на основе не классов, а заранее реализованных объектов-прототипов [5]. При реализации синтаксического анализатора на JavaScript использовалась платформа Node.js, позволяющая создавать кроссплатформенные высокопроизводительные приложения.

C# – это объектно-ориентированный язык программирования от Microsoft, который в значительной степени фокусируется на объединении вычислительной мощности C++ с простотой программирования Visual Basic. C# основан на C++ и включает в себя функции, аналогичные функциям Java. Это смесь C и C++. C# обычно используется для серверных и десктопных приложений, разработки веб-сайтов и игр [6]. Реализация синтаксического анализатора осуществлялась на платформе .Net, являющейся кроссплатформенной экосистемой разработки. Она обеспечивает производительность, безопасность и надежность, а также удобную разработку благодаря высокопроизводительной среде выполнения и удобным встроенным инструментам [7].

При создании синтаксического анализатора требуется первым делом определить структуру будущей программы, ее архитектуру. Тут сразу же можно столкнуться с различиями в этих двух языках.

На языке C# можно реализовать анализатор с помощью классов и интерфейсов, позволяющих сделать абстракцию над классами, содержащую только ту часть реализации, которой будут открыто пользоваться другие классы. Это позволит обезопасить внутреннее состояние классов и инкапсулировать данные. Для реализации синтаксического анализатора в программе были созданы такие классы, как Lexeme, Lexer, Parser и классы-узлы синтаксического дерева. Класс Lexeme представляет собой лексему и хранит в себе тип и значение каждой единицы языка. Класс Lexer является лексическим анализатором, он разбивает входную сроку на лексемы и проверяет, соответствуют ли они грамматике языка. В этом классе есть метод Next, по очереди выдающий лексемы входной строки. Классы-узлы синтаксического дерева представляют собой базовый абстрактный класс и остальные классы, наследуемые от него. Все классы-наследники реализуют какое-либо правило грамматики языка, устанавливающее в каком порядке, в зависимости от полученной на вход лексемы, в стек будет положено то или иное правило грамматики. Класс Parser реализует синтаксический анализатор. Этот класс содержит в себе стек магазинного автомата, позволяющий хранить в себе символы грамматики для сопоставления с входной строкой. С помощью класса Lexer синтаксический анализатор по порядку получает лексемы и проверяет правильность их следования, строя синтаксическое дерево. В классах Lexer и Parser реализована опциональная обработка ошибок.

Реализация на языке JavaScript представляет собой совершенно другую программу. Для реализации синтаксического анализатора на этом языке удобно использовать функции, представля-

ющие собой объект, ссылающиеся на глобальный объект-прототип. В программе были созданы такие функции, как Lexeme, Lexer и Parser, а также функции-узлы синтаксического дерева. В этих функциях реализована та же логика, что и в классах в программе на C#, т.е. функции Lexeme и Lexer также возвращают объект, а Parser хранит стек и составляет синтаксическое дерево, проверяя синтаксис входной строки на основе функций-узлов синтаксического дерева.

Рассмотрим, в чем состоит отличие этих двух реализаций. Во-первых, в JavaScript отсутствует строгая типизация данных. Это означает, что в этом языке, в отличие от C#, объекты не имеют своего типа. Эта особенность JavaScript позволяет сделать разработку синтаксического анализатора более быстрой, простой и гибкой. С другой стороны, это может быть причиной для множества ошибок, которые будут заметны только во время выполнения программы. Программа на C# более сложная и длительная в реализации, но является более безопасной с точки зрения инкапсулирования данных и ошибок типа, которые можно будет заметить уже на этапе компиляции. Также программа на C# будет лучше структурирована, а код благодаря строгой типизации будет являться более понятным. Это же относится и к обработке ошибок, ведь исключения в этом языке строго типизированы и проверяются компилятором.

Во-вторых, как уже было сказано выше, в JavaScript нет классов в классическом их понимании. Хоть они и реализованы в новом стандарте языка ES6, внутри они являются объектами, а конструкторы и методы класса – те же функции [8]. Для программы, реализующей синтаксический анализатор, были использованы чистые функции. Это позволило значительно ускорить разработку, а также сделать программу более гибкой за счет разнообразного контекста использования функций в JavaScript. Также можно добавить, что JavaScript – это интерпретируемый язык, а C# – компилируемый. Благодаря этому программа на C# будет быстрее работать и показывать лучшую производительность в высоконагруженных программах [9].

В заключение можно сказать, что каждый из этих подходов к разработке имеет свои преимущества. Разработка синтаксического анализатора на JavaScript хороша в условиях быстрой разработки и при необходимости гибкости программы, а на C# – при необходимости обеспечения надежности и высокой производительности. Из сказанного можно сделать вывод, что при разработке синтаксического анализатора для языка программирования использование C# будет лучшим решением, поскольку инструменты языка и его характеристики больше подходят для работы с большими входными данными при программировании.

Список литературы

1. Beatty J. C. On the relationship between LL(1) and LR(1) grammars // Journal of the ACM. 1982. С. 1008–1022. URL: semanticscholar.org>

2. Хопкрофт Д. Э., Мотвани Р., Джеффри У. Д. Введение в теорию автоматов, языков и вычислений : пер. с англ. 2-е изд. М. : Вильямс, 2008. 528 с.

3. Ахо А. В., Лам М. С., Сети Р., Ульман Д. Д. Компиляторы: принципы, технологии и инструменты. 2-е изд. М. : Вильямс, 2008. 1186 с.

4. Types of Parsers in Compiler Design. URL: https://www.geeksforgeeks.org (дата обращения: 13.10.2024).

5. Крокфорд Д. JavaScript: сильные стороны. СПб. : Питер, 2012. 176 с.

6. Mastering C#: A Beginner’s Guide / Edited by Sufyan bin Uzayr. CRC Press, 2022. 308 p. doi: 1201/9781

7. Документация по основам .NET. URL: https://learn.microsoft.com/ru (дата обращения: 19.10.2024).

8. Мавлянов А. Н. У. Эффективная реализация традиционных концепций ООП с использованием прототипного программирования // E-Scio. 2020. № 10. С. 398–407.

9. Пилипенко А. Обзор интерпретации и компиляции в виртуальных машинах // Компьютерные инструменты в образовании. 2012. № 3. С. 3–15.

Информация об авторах

Смирнов Илья Алексеевич, студент, Пензенский государственный университет.

Пичушкина Маргарита Александровна, студентка, Пензенский государственный университет.

Дорофеева Ольга Станиславовна, кандидат технических наук, доцент, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 681.3

КЛАССИФИКАЦИЯ ОПОВЕЩЕНИЙ И ИХ РОЛЬ В СОВРЕМЕННЫХ ПРИЛОЖЕНИЯХ

А. Е. Смирнова

Пензенский государственный университет, Пенза, Россия

angelina422smirnova@yandex.ru

Аннотация. Классификация оповещений и их роль в современных приложениях становятся все более актуальными в условиях быстрого развития технологий и увеличения объема информации. Рассматриваются различные типы оповещений, такие как системные уведомления, push-уведомления, оповещения через API и др. Анализируются оповещения по типу решения их задач. Упоминается важность четкой классификации оповещений для повышения эффективности коммуникации между приложением и пользователем, а также для минимизации раздражения пользователя. Кроме того, рассматриваются практики внедрения оповещений в приложения (с описанием взаимодействия пользователя с ними). Подчеркивается, что роль уведомлений велика и использование оповещений способно значительно улучшить функциональность современных приложений.

Ключевые слова: оповещение, классификация, уведомление, современные приложения, интерактивность, мобильное приложение, эффективность оповещений

Благодарности: статья подготовлена под научным руководством кандидата технических наук, доцента С. В. Шибанова.

Для цитирования: Смирнова А. Е. Классификация оповещений и их роль в современных приложениях // Вестник Пензенского государственного университета. 2024. № 4. С. 122–125.

При реализации каждого приложения предусматривают систему уведомлений, которая является средством информирования пользователя о каких-либо действиях. Правильное использование выбранных уведомлений позволяет разработчикам создавать эффективную работу приложения и улучшает качество взаимосвязи с конечным пользователем.

Оповещения могут решать различные задачи. В зависимости от задач можно классифицировать следующие оповещения [1]:

1. Оповещения, предоставляющие пользователю информацию от приложения о его состоянии или данных. Примером может стать оповещение об успешной авторизации.

2. Оповещения, сообщающие о необходимости принять меры. Например, уведомление об обновлении системы.

3. Оповещения, появляющиеся при ошибках или исключениях. Такие оповещения могут возникать при неверном вводе данных.

4. Оповещения, подтверждающие действие пользователя. Чаще всего такой тип уведомлений используют при подтверждении удаления.

5. Оповещения, которые информируют о событиях на уровне системы. Например, завершение фоновых задач.

6. Оповещения, сообщающие о статусе текущей задачи или процесса. Примером может стать оповещение, возникающее при загрузке или скачивании.

7. Оповещения, срабатывающие автоматически при наступлении необходимого условия. Такие оповещения можно называть триггерными.

Определив требования и цели разрабатываемого приложения, можно выбрать список необходимых оповещений. Как правило, в одном приложении может использоваться сразу несколько типов уведомлений.

Также оповещения можно классифицировать по их типу при разработке. В табл.1 представлены названия и описания оповещений.

Таблица 1

Виды оповещений в программировании

Название		Описание	Пример
1	Системные уведомления	Уведомления операционной системы о событиях	Фоновые задачи, обновления
2	Уведомления в приложениях	Оповещения об активности внутри приложения	Уведомление о новых сообщениях или появившихся событиях
3	Email-уведомления	Оповещения, отправляемые на электронную почту	Сброс пароля или подтверждение регистрации
4	Push-уведомления	Уведомления, отправляемые на мобильные устройства или браузеры, даже когда приложение неактивно [2]	Уведомление о начавшемся звонке из мессенджера
5	Логирование	Запись сообщений в лог для отслеживания состояния приложения и выявления ошибок	При появлении багов во время звонка
6	Оповещения через API	Коммуникация между сервисами, где один сервис может уведомлять другой о событиях через вызовы API	Уведомление об успешном получении данных
7	Чат-боты	Оповещения в мессенджерах с помощью ботов	Сообщения о различных событиях
8	Визуальные уведомления	Используются в графических интерфейсах для взаимодействия с пользователем	Всплывающие окна или алерты

Сравнение оповещений разных типов можно проводить по различным критериям:

– скорость доставки;

– канал передачи;

– аудитория;

– удобство получения;

– сложность разработки;

– частота использования;

– уровень вовлеченности.

Сравнив по этим критериям, можно определить, какой тип оповещений наиболее подходит для конкретной ситуации.

Важным аспектом уведомлений является «интерактивное» взаимодействие с ними. Благодаря быстрому развитию различных операционных систем уведомления перестали служить только для получения информации. Так, например, в системах IOS и Android большинство уведомле-

ний от системных приложений получили новые функции, благодаря которым можно совершить какое-либо действие, при этом не запуская нужное приложение [3].

Рассмотрим приложение «Часы» на IOS. В данном приложении есть функция таймера, по истечении которого пользователь получает уведомление (рис. 1).

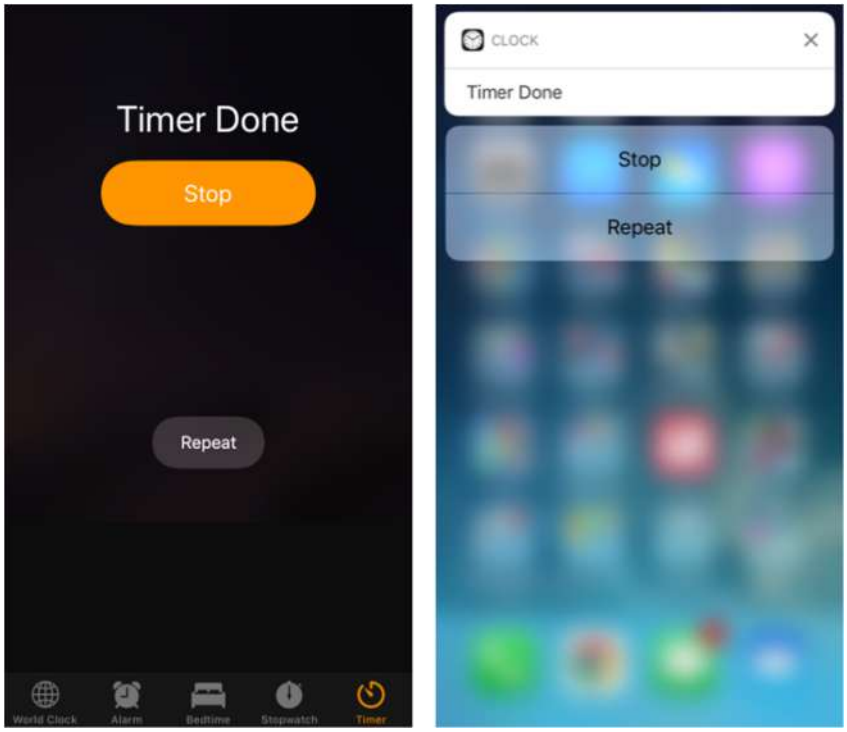


Рис. 1. Отображение работы уведомлений таймера

Однако, кроме получения информации о том, что прошел заданный промежуток времени, пользователь также может запустить таймер заново без необходимости открывать приложение.

Также активное взаимодействие уведомлений и их получателя показывают различные мессенджеры: Вконтакте, Telegram, Platrum Chat и др. При получении уведомления о сообщении в одном из них пользователь может прямо через него выбрать один из нескольких вариантов взаимодействия. Так, например, можно открыть диалоговое окно для набора ответного сообщения, отправить заранее заготовленный короткий ответ или вовсе отключить уведомления о сообщениях, причем это работает не только на мобильных операционных системах, но и на стационарных компьютерах.

В уведомлении о новом сообщении из приложения Вконтакте можно выбрать одно из действий, не заходя в приложение:

– ответить;

– ответить быстрой реакцией;

– прочитано;

– не беспокоить 8 ч.

Данные функции улучшают работу приложения, добавляя удобства для пользователя в использовании.

Еще одним примером «интерактивных» уведомлений служит решение от «Yandex Go». Это приложение для заказа такси и доставки различных товаров. Ранее при заказе такси пользователь получал лишь два типа уведомлений – принятие заказа и прибытие водителя [4]. Однако с одним из последних обновлений разработчики внесли глобальные изменения (рис. 2).

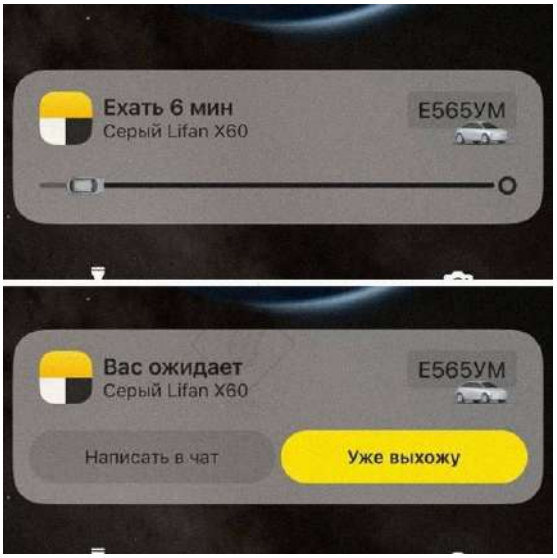


Рис. 2. Уведомления в Yandex Go

Теперь пользователь видит гораздо больше информации в уведомлении от приложения:

– марка, цвет и госномер автомобиля;

– отправка информации водителю;

– примерное время до прибытия автомобиля к заказчику, а затем время до конца поездки;

– сумма и статус оплаты за оказанные услуги.

В современном мире, где все процессы происходят все быстрее, роль оповещений как никогда велика. Благодаря ним пользователи различных систем, программного обеспечения и приложения могут получать корректную и своевременную информацию о том или ином событии и, как мы теперь убедились, даже повлиять на происходящий процесс. Сейчас уведомления есть у любого мобильного приложения и у многих веб-интерфейсов. Их использование помогает сокращать количество ошибок и повышать уровень обслуживания при использовании того или иного программного обеспечения.

Список литературы

1. Wexler J. Designing Effective Notifications. 2010. P. 21–27.

2. Аксенов К. А., Сухарев О. С. Проектирование систем информирования клиентов. 2023. С. 11–12.

3. Федотенко М. А. Разработка мобильных приложений. Первые шаги. М. : Лаборатория знаний, 2019. С. 112–123.

4. Яндекс начал выводить статус такси на экран блокировки. URL: https://rb.ru

Информация об авторе

Смирнова Ангелина Евгеньевна, студентка, Пензенский государственный университет.

Автор заявляет об отсутствии конфликта интересов.

УДК 004.75

ВАЛИДАЦИЯ ПРОТОКОЛА TCP С ИСПОЛЬЗОВАНИЕМ NK-ПОДОБНОГО АВТОМАТА

В. А. Стежка

АО «Радиозавод», Пенза, Россия

adevelgio@gmail.com

Аннотация. Описан способ подтверждения требований надежности, отказоустойчивости и непротиворечивости реализации протокола TCP. Приведены модификации NK-автомата Кауфмана, которые представляют интерес с точки зрения генерации информационных пакетов TCP. Разработано программное средство, позволяющее провести эксперимент. Проанализированы результаты.

Ключевые слова: протокол передачи данных, валидация, NK-автомат, аттрактор, приспособленность, генерация

Для цитирования: Стежка В. А. Валидация протокола TCP с использованием NK-подобного автомата // Вестник Пензенского государственного университета. 2024. № 4. С. 126–130.

Введение

Окружающая нас цифровая действительность предполагает огромное количество электронных устройств, взаимодействующих друг с другом. Множество информационных потоков, распространяемых в такой среде, обязано подчиняться строгим правилам, обеспечивающим безопасность и надежность передаваемой информации. Набор соглашений, определяющих характер информационного взаимодействия, содержится в протоколе передачи данных. Например, TCP – протокол управления передачей, первое описание RFC-675 получил в 1974 г., именно в тексте этого документа впервые появился термин «Internet», что дополнительно подчеркивает их неразрывную связь. За прошедшие 50 лет текст протокола несколько раз переписывался (RFC-793, RFC-9293), уточнялся и дополнялся (RFC-879, RFC-1122, RFC-2873).

Методом доказательства пригодности информационного протокола для использования является его валидация. Протокол TCP регламентирует последовательность взаимодействия двух узлов сети между собой, и эта последовательность имеет строго определенный набор команд. Для валидации TCP ранее уже использовались модели конечных автоматов [1], графов переходов или сети Петри [2].

Цель работы: дать описание способа валидации протокола TCP модифицированным NK-автоматом Кауфмана [3] в ядре ОС Линукс версии 6.5.0, описанной стандартами RFC-793, RFC-1122 и RFC-2001.

Метод исследования

Автономная бинарная сеть NK-автомата использует случайным образом распределенные функции и связи (K). Каждый элемент (N) реализует булевую функцию, позволяющую вычислять его текущее значение. Каждый элемент в бинарной сети может быть в активном (1) или неактивном состоянии (0). Функционирование автомата заключается в смене состояния его элементов

с течением времени по определенным правилам. В результате некоторого количества циклов работы автомата неизбежно возникнет ситуация, в которой состояние всех элементов уже встречалось ранее, такое событие называется аттрактором. Важной характеристикой является количество циклов, необходимое для повторения состояния автомата, это называется длиной аттрактора. Достижение аттрактора в ходе текущего эксперимента будет являться сигналом для генерации TCP пакета.

Модель NK-автомата связана с теорией эволюции живых организмов, поэтому в ней определены правила мутации с целью достижения наиболее приспособленного варианта. Для качественной оценки вариантов NK-автомата используется понятие «приспособленность» (*fitness*). Приспособленность NK-автомата – среднее значение приспособленности всех элементов, вычисляется по формуле

$$f(x) = \frac{\sum_{i=1}^{N} f_i(x)}{N},$$

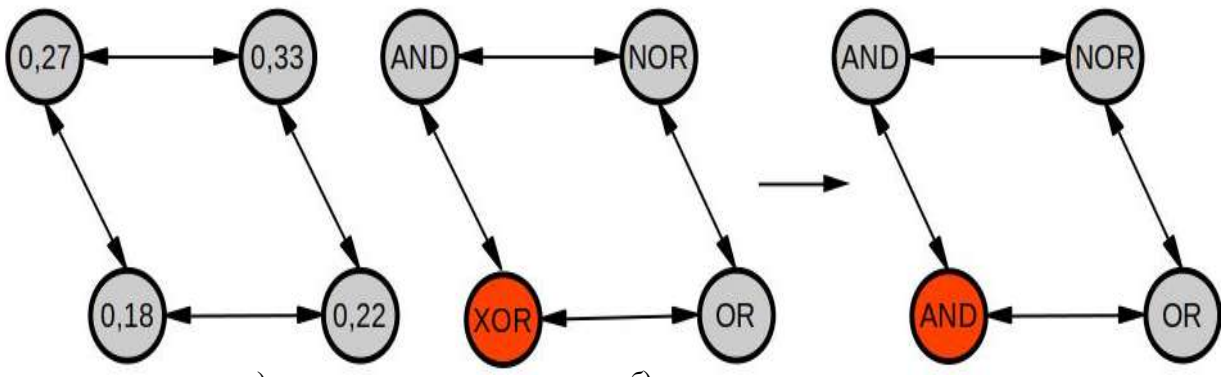
где N – количество элементов; $f(x)$, $f_i(x)$ – приспособленность всего автомата, каждого элемента; x – текущее состояние.

Вариант с наибольшим значением приспособленности будет называться «локальным максимумом» (*local maximum*) и, как самый приспособленный, будет являться наиболее подходящим прародителем для последующих изменений.

Следующие модификации NK-автомата Кауфмана представляют интерес в тематике текущей статьи:

– в NK/W-автомате [4] каждый элемент первоначально получает в соответствие значение приспособленности в диапазоне от 0.0 до 1.0, чтобы в сумме приспособленность всех элементов равнялась 1 (рис. 1,*а*). Если элемент активен, он вносит свой вклад в общую приспособленность текущего варианта NK-автомата. Вариант с наибольшей приспособленностью выбирается в качестве прародителя для последующих мутаций. Также в данной модификации может быть выбран «сильно взвешенный» элемент, значение приспособленности которого значительно превышает приспособленности всех остальных элементов, вместе взятых;

– RMNK-автомат [5] являются результатом комбинации случайных булевых сетей (RBN) и NK-автоматов. Модификации автомата меняют логические функции некоторых элементов либо меняют связанность с другими элементами. Изменение автомата происходит при достижении аттрактора, а выбор конкретной модификации принимается на основе поиска варианта с наибольшей приспособленностью (рис. 1,*б*);



а) *б*)

Рис. 1. NK-автоматы: *а* – NK/W-автомат, *б* – RMNK-автомат

– NK/A-автомат [4] добавляет параметр A, определяющий количество аффекторов, которые в приоритетном порядке соединяются с остальными элементами. Таким образом вклад каждого элемента в общую приспособленность зависит от его внутреннего состояния и от состояния аффектора. Особенность NK/A-автомата заключается в том, что в модели будет не более 2^A локальных максимумов приспособленности, а по сравнению с базовым NK-автоматом количество шагов для достижения локального максимума уменьшается по мере увеличения числа аффекторных элементов;

– NK-подобный автомат [6] предложенный Е. А. Кольчугиной, добавляет управляющий элемент S, который определяет порядок вычисления функциональных элементов, также добавляются замедляющие элементы, хранящие состояние связанных с ними элементов в предыдущий момент времени. Все функциональные элементы соединены только с управляющим, а замедлители – только с функциональными элементами. Благодаря этим изменениям автомат приобретает свойства конкурентного доступа к элементам автомата при одновременно выполняемых вычислениях даже в последовательном режим функционирования.

Для подтверждения требований надежности, отказоустойчивости и непротиворечивости протокола TCP будем использовать TCP-NK-автомат, работа которого будет направлена на генерацию TCP пакетов, передаваемых как в адрес «базового» TCP сервера (POSIX API) так и в адрес сервера, написанного с помощью библиотеки Qt (QTcpServer). Генерацию пакета будет обеспечивать автомат из восьми элементов ($N = 8$) (табл. 1).

Таблица 1

Элементы TCP-NK-автомата

Номер	Значение	Комментарий
1	RST	RFC-793, оборвать соединение
2	FIN	RFC-793, указание на завершение соединения
3	SYN	RFC-793, синхронизация номеров последовательности
4	PUSH	RFC-793, проталкивание данных, наличие данных определяется DAT
5	ACK	RFC-793, номер подтверждения
6	URG	RFC-793, указатель важности
7	WND	Признак изменения размера «окна»
8	DAT	Признак наличия данных в пакете

Количество связей между элементами (K) будем выбирать в диапазоне от 1 до 7. Дополнительно проверим работу автомата с аффекторными элементами (A), количество которых будем выбирать в диапазоне $0 < A < N$. Каждому элементу будет соответствовать выбранная случайным образом булевая функция из набора: И, ИЛИ, исключающее ИЛИ, И-НЕ, ИЛИ-НЕ, НЕ-ИЛИ. При достижении аттрактора будет происходить формирование TCP пакета, в который каждый активный элемент будет добавлять свои «свойства». Также при достижении аттрактора будем модифицировать автомат в соответствии с одной из следующих стратегий (табл. 2).

Таблица 2

Стратегии модификации TCP-NK-автомата

Номер	Название	Описание
1	RANDOM	Для случайного узла меняется булевая функция или степень эпистатичности
2	FITTEST	Для каждого узла поочередно меняется булевая функция или степень эпистатичности, рассчитывается суммарная приспособленность, выбирается вариант с локальным максимумом

Для оценки приспособленности NK-автомата между всеми узлами случайным образом распределим значение веса (w), таким образом, чтобы $w_1 + w_2 + ...w_8 = 1{,}0$. Активный элемент будет давать вклад в общее значение приспособленности, а неактивный нет. Вариант с наибольшем значением W и будет считаться локальным максимумом, соответственно ему будет отдаваться предпочтении в «FITTEST» стратегии.

Выберем последовательный порядок вычисления значений каждого элемента на конкретном такте работы автомата. Состояние автомата можем выразить в виде набора состояний каждого узла по формуле

$$P(x) = \{P_1(x), P_2(x), P_3(x), P_4(x), P_5(x), P_6(x), P_7(x), P_8(x)\},$$

где $P(x)$ – состояние TCP-NK-автомата; $P_n(x)$ – состояние конкретного элемента.

Так как каждый элемент в автомате в конкретный момент времени находится в активном или неактивном состоянии, то, например, при последовательном расчете возникает ситуация, что элемент с порядковым номером 1 для определения своего состояния вынужден использовать состояния остальных элементов с предыдущего такта работы автомата. Элемент под номером 2 уже сможет обратиться к обновленному значению элемента с номером 1, а все последующие по-прежнему будут содержать «старые» значения. Таким образом при продолжении вычислений произойдет постепенное обновление состояний всех элементов и самый последний, 8-й элемент, будет использовать только «новые» состояния каждого из связанных элементов. Для того, чтобы убрать зависимость от порядка вычислений, добавим в работу TCP-NK-автомата замедлители, которые будут сохранять «старое» значение каждого элемента на текущем такте работы автомата. Таким образом будет достигнут эффект одновременного «срабатывания» всех элементов.

Для подтверждения правильности и стабильности реализации TCP в ОС Ubuntu 22.04 была разработана отдельная утилита, запуск которой проводился совместно с TCP-NK-автоматом. Утилита подключается к серверу, генерирует случайный массив байт, передает его на сервер, ждет ответа и сравнивает полученные данные от сервера с отправленными. Если полученные данные соответствуют, то фиксируется успешное выполнение и происходит отключение от сервера с повторным повторением процедуры. Если во время подключения, передачи или получения данных возникает ошибка, происходит фиксация этого события, и дальше процедура подключения и передачи повторяется. В процессе верификации стека TCP в ОС Ubuntu было одновременно запущено несколько таких утилит.

Результаты работы различных конфигураций TCP-NK-автомата на 100.000 поколений (каждое поколение автомата формировало информационный пакет TCP) показали стабильность, отказоустойчивость и корректность реализации протокола TCP в ОС Ubuntu 22.04. Среднее значений длин достижения аттракторов приведено на рис. 2.

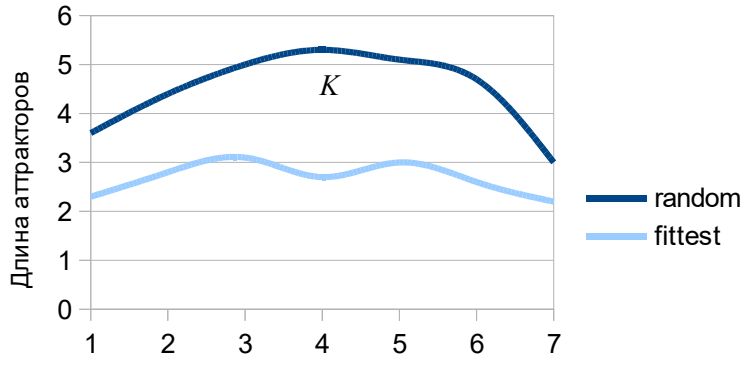


Рис. 2. Среднее значений длин достижения аттракторов

В чем преимущество работы случайно сгенерированного автомата по отношению к технике тестирования протокола «мусорными данными» (fuzz testing)? Фаззинг схож с работой TCP-NK-автомата со случайной стратегией модификации, а в случае выбора наиболее приспособленных вариантов появляется возможность благодаря параметру – коэффициент веса смещать акцент в сторону наиболее «важных» свойств пакетов, например ACK или SYN.

Протокол TCP имеет небольшое количество свойств, использование «базовых» булевых функций И и ИЛИ недостаточно, все узлы автомата будут быстро переходить в неактивное состояние, поэтому необходимо добавить исключающие булевые функции, такие как И-НЕ, ИЛИ-НЕ, НЕ-ИЛИ, восстанавливающие активное состояние элементов.

Заключение

Разработанный TCP-NK-автомат продемонстрировал, что модель NK-автомата применима для валидации протокола TCP. Так как полезная работа автомата заключается в генерации TCP пакетов, стратегия с выбором наиболее приспособленных вариантов изменения автомата получает преимущества по сравнению со случайной, так как уменьшается количество шагов для достижения аттрактора. В ходе работы 100.000 поколений TCP-NK-автомата не было выявлено сбоев и отказов в работе стека TCP, реализованного ОС Ubuntu 22.04.

В статье рассматривался протокол TCP, но с помощью модели NK-автомата возможно проводить валидацию и других информационных протоколов, а сочетание различных модификаций позволяет создать конфигурацию с необходимыми свойствами.

Список литературы

1. Smith M. A. S. Formal Verification of TCP and T/TCP. MIT, 1997. 424 p. URL: semarticscholar.org>

2. Han B., Billington J. Validating TCP Connection Management. University of South Australia, 2002. 9 p. URL: researchgate.net>

3. Kauffman S. A. The Origin of Order: Self-Organization and Selection in Evolution. New York : Oxford University Press, 1993. 710 p.

4. Solow D., Burnetas A., Roeder T., Greenspan N. Evolutionary Consequences of Selected Locus-Specific Variations in Epistasis and Fitness Contribution in Kaufman's NK Model. Cleveland, USA : Case Western Reserve University, 2001. 20 p.

5. Bull L. Coevolving Boolean and Multi-Valued Regulatory Networks. Bristol, UK : Computer Science Research Centre. University of the West of England, 2023. 21 p.

6. Кольчугина Е.А. Исследование свойств цифровых организмов с помощью NK-подобных автоматов // Вестник компьютерных и информационных технологий. 2011. № 11. С. 20–24.

Информация об авторе

Стежка Владимир Андреевич, начальник программного отдела, АО «Радиозавод» (г. Пенза).

Автор заявляет об отсутствии конфликта интересов.

УДК 004.43

СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ ПРИ ВЫПОЛНЕНИИ СОРТИРОВКИ МЕТОДОМ ПУЗЫРЬКА НА ЯЗЫКАХ ПРОГРАММИРОВАНИЯ C++ И PYTHON

М. Д. Сычёв[1], К. Д. Трунова[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1] matveika.2004@mail.ru
[2] trunova.kseniya2004@gmail.com

Аннотация. Анализируется скорость работы языков программирования Python и C++ на примере выполнения пузырьковой сортировки. Выясняются особенности этих языков и причины ощутимой разницы во времени выполнения одинаковой задачи.

Ключевые слова: Python, C++, язык программирования, сравнение, пузырьковая сортировка, скорость работы, время выполнения, интерпретируемый язык, компилируемый язык, сборка мусора, управление памятью

Для цитирования: Сычёв М. Д., Трунова К. Д. Сравнение производительности при выполнении сортировки методом пузырька на языках программирования C++ и Python // Вестник Пензенского государственного университета. 2024. № 4. С. 131–133.

На данный момент в мире C++ и Python являются одними из самых известных и часто используемых языков программирования (ЯП). Оба этих языка очень эффективны для определенных задач и являются востребованными на рынке. В данной статье рассматривается вопрос сравнения скорости работы этих языков, по которой Python значительно проигрывает C++. Для того, чтобы самостоятельно убедиться в большой разнице скорости их работы, а также разобраться, с чем это связано, на двух языках был написан код пузырьковой сортировки [1].

Тестирование началось с 1000 элементов. Время выполнения сортировки на языках Python и C++ соответственно: 0,0292 с; 0,001 с. Очевидно, что на данный момент оба языка справляются почти мгновенно, но уже наблюдается отставание Python.

Далее число элементов было увеличено до 5000. Результаты оказались следующими: 0,8398 с; 0,016 с. В этот раз Python’у потребовалась почти секунда, а C++ меньше одной десятой секунды.

Еще раз увеличим число элементов, в этот раз до 20 000. Результаты: 12,4412 с; 0,496 с. На примере с большим количеством элементов разница в скорости работы C++ и Python становится очевидной, а именно: C++ справился в 25 раз быстрее, чем Python. Если представить полученные результаты в графическом виде, то получится следующее (рис. 1).

По графику видно, что прирост времени выполнения задач у C++ почти незначительный, в то время как у Python время выполнения сортировки кратно увеличивается.

Есть несколько причин такой большой разницы в скорости работы этих двух языков программирования.

Во-первых, Python является интерпретируемым языком [2]. Это означает, что для него требуется отдельная программа – интерпретатор (python, он же CPython), которая построчно читает и исполняет код, без его предварительной компиляции в машинный код. Это дает следующие пре-

имущества: большую гибкость, динамическую типизацию и меньший размер программы, а также возможность выполнять код на множестве различных платформ.

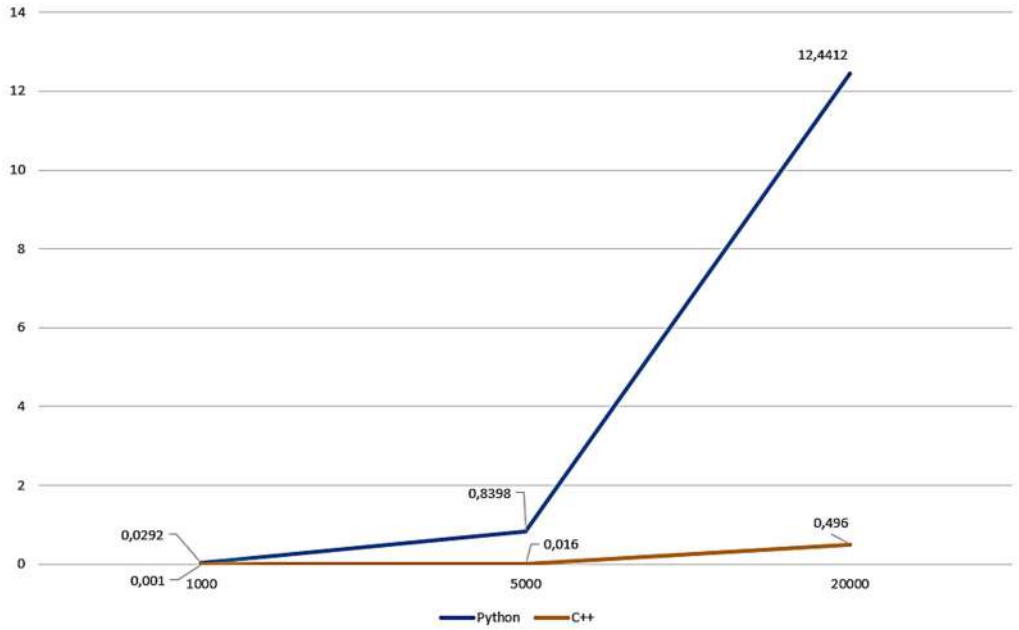


Рис. 1. Динамика изменения скорости работы сравниваемых ЯП

В свою очередь, C++ является компилируемым языком [3], т.е. программа-компилятор сразу считывает весь исходный код и преобразует его в машинный код, с которым далее напрямую может работать процессор. Преимуществом этого способа является скорость работы, но недостатками — долгий процесс сборки программы, а также зависимость от платформы (например, один и тот же код может работать под ОС Windows, но не работать под Linux).

Таким образом, на C++ код долго подготавливается, но затем быстро выполняется, а на Python процесс выполнения начинается сразу же, однако длится дольше из-за последовательной обработки каждой строки кода.

Во-вторых, на производительность влияет наличие механизмов автоматической сборки мусора и управления памятью [2]. В Python не нужно отслеживать, когда используется память и когда она больше не нужна, система делает это автоматически. В C++ же задачей программиста является отслеживание памяти и явное указание на то, когда ее надо очистить [3]. Также, если во время выполнения кода на Python попытаться прочитать данные за пределами структуры памяти, будет получена ошибка. В C++ же можно считывать данные за пределами массива или перед его началом, что может привести к утечкам памяти и различным исключениям.

Как в языке Python, автоматизация этих процессов упрощает написание программ и их отладку, однако это достигается за счет увеличения использования памяти и снижения скорости выполнения, ведь системе приходится делать гораздо больше проверок. В свою очередь, C++ позволяет выполнять код быстрее, но вопрос сборки мусора и управления памятью возлагается на программиста и делает написание кода более трудоемким процессом.

Именно поэтому выполнение программы на Python занимает больше времени, чем на C++. Однако это не значит, что Python является хуже C++. У обоих языков есть свои достоинства и недостатки. Каждый разработчик выбирает язык в зависимости от задач, поставленных перед ним. Преимущества Python заключаются в простоте и быстроте написания кода, кроссплатформенности, богатом наборе различных библиотек. C++ при этом обладает высокой производительностью и позволяет вести более глубокую и продуманную разработку.

Список литературы

1. Вирт Н. Алгоритмы и структуры данных. М. : ДМК Пресс, 2010. 272 с.
2. Лутц М. Изучаем Python. 5-е изд. СПб. : Диалектика, 2019. Т. 1. 832 с.
3. Страуструп Б. Язык программирования C++. М. : Бином, 2006. 1104 с.

Информация об авторах

Сычёв Матвей Дмитриевич, студент, Пензенский государственный университет.

Трунова Ксения Дмитриевна, студентка, Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.4

ВЛИЯНИЯ ТЕХНОЛОГИЙ КОНТЕЙНЕРИЗАЦИИ ПРИЛОЖЕНИЙ НА СКОРОСТЬ ВЫПОЛНЕНИЯ

Д. В. Холодков[1], С. А. Зинкин[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1] danila.holodkov@mail.ru
[2] zsa49@yandex.ru

Аннотация. Рассматривается влияние контейнеризации с использованием Docker на скорость работы программного обеспечения внутри контейнеров. Несмотря на все удобства, предлагаемые разработчикам технологиями контейнеризации, существуют некоторые ограничения и издержки, связанные с использованием дополнительной программной прослойки между программным обеспечением и вычислительной машиной, в частности замедление работы программы. Основное внимание уделяется сравнению производительности программ, написанных на языке программирования Java, развернутых в контейнерах и при традиционном способе установки.

Ключевые слова: контейнеризация, Docker, время выполнения, среда выполнения, виртуализация, сравнительный анализ, Java

Для цитирования: Холодков Д. В., Зинкин С. А. Влияния технологий контейнеризации приложений на скорость выполнения // Вестник Пензенского государственного университета. 2024. № 4. С. 134–136.

Контейнеризация программ позволяет упаковывать приложения вместе с их зависимостями в изолированные контейнеры и изолировать выполняемую программу от программ на исполняемой машине [1]. Также это позволяет не беспокоиться об установке вспомогательных программ, необходимых для работы приложения, потому что появляется возможность поставлять все необходимое уже в контейнере. Существует несколько различных приложений, позволяющих реализовать контейнеризацию приложения. Самым распространенным является Docker, поэтому в данной статье было принято решение использовать Docker во время анализа и тестирования.

При всех плюсах контейнеризации у разработчиков, которые решили применять в работе технологии контейнеризации приложений, могут возникнуть ситуации, в которых использование контейнеризации приложения принесет больше вреда, чем пользы. Одним из минусов использования контейнеров является снижение производительности. Это самый очевидный минус, который может ожидать разработчика. Сам по себе контейнер в большинстве случаев является виртуальной машиной с облегченной операционной системой, на которой устанавливаются все необходимые программы для работы приложения и само приложение. Получается, что наше приложение упаковывается внутрь другого приложения со всеми зависимостями и изолировано от программ и процессов основной операционной системы. Таким образом, между приложением и ядром процессора появляется еще один, дополнительный, программный слой, что неминуемо приводит к замедлению работы приложения [2].

Однако нельзя сказать, что все приложения будут одинаково замедлены, иначе можно было бы легко учитывать это при разработке. Если бы все приложения одинаково замедлялись на

5 %, то можно было бы выделить на 5 % больше аппаратных ресурсов, но нельзя предугадать насколько надо увеличить вычислительные мощности на этапе разработки, чтобы их было достаточно и при этом не было лишних затрат. От типа операции и оптимизации этого типа операции в контейнере будет зависеть, насколько она будет замедлена. Рассмотрим два типа операций: арифметическую операцию и операцию чтения и записи в файл. При этом не имеет значения сложность выполняемой задачи. Мы можем поставить такие условия задачи, что программа будет выполняться 5 ч, а можем поставить задачу на 5 с. Имеет значение только разница выполнения одной и той же задачи в условиях применения контейнеризации и без ее применения.

Для оценки замедления арифметической операции была написана программа на языке Java, которая высчитывает интеграл методом трапеции с шагом 0.0000001. Вычисляемый интеграл

$$\int_{1000}^{1} \frac{1}{x} dx.$$

Для выполнения тестирования было несколько вариантов выбора языка программирования, однако было принято решение остановиться на Java как на одном из самых распространенных языков программирования [3], благодаря чему данное тестирование будет полезно большему числу разработчиков.

Для оценки замедления операции чтения и записи в файл была написана программа, выполняющая последовательно чтение текстового файла весом 100 МБ и запись в новый файл тех же данных 40 раз. Снимки экрана с выполнением программ представлены на рис. 1–4. Результаты тестов представлены в табл. 1.

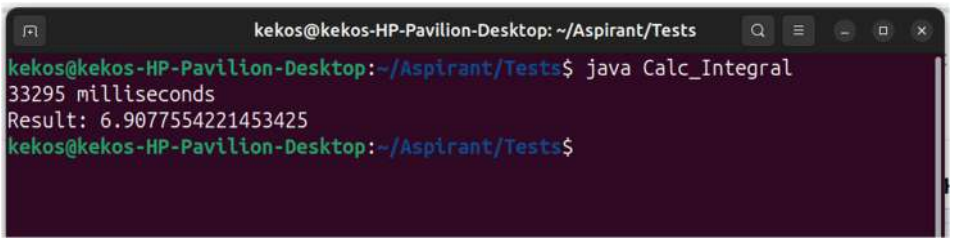


Рис. 1. Результат выполнения программы вычисления интеграла без контейнера

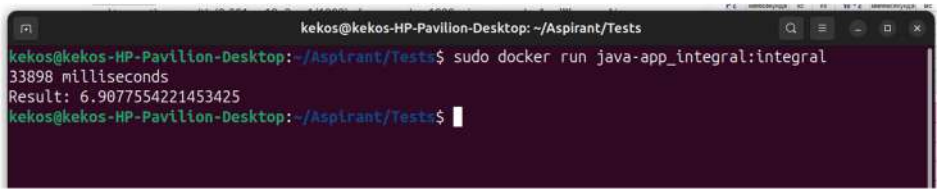


Рис. 2. Результат выполнения программы вычисления интеграла в контейнере

Результаты тестирования программы, выполняющей вычисление интеграла методом трапеции, показывают минимальную разницу во времени выполнения.

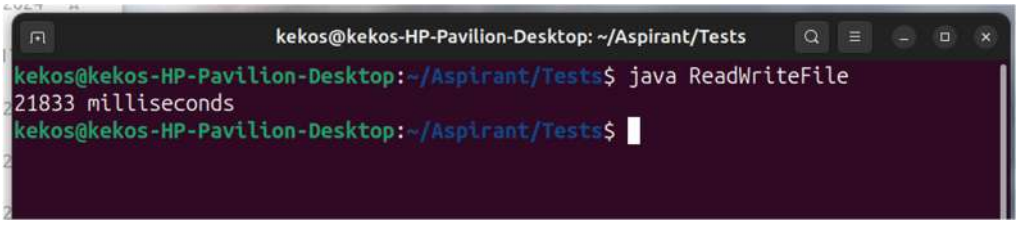


Рис. 3. Результат выполнения программы чтения и записи в файл без контейнера

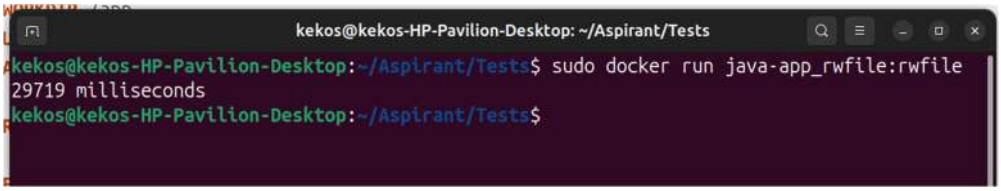


Рис. 4. Результат выполнения программы чтения и записи в файл в контейнере

Результаты тестирования программы, выполняющей чтение и запись в файл, показывают значительную разницу во времени выполнения. Без контейнера программа выполняется на 26 % быстрее. При разработке приложений, работающих с файлами, следует проанализировать целесообразность применения контейнеризации.

Таблица 1

Результаты тестов

Программа	Время выполнения без контейнера, мс	Время выполнения в контейнере, мс
Вычисление интеграла	33 295	33 898
Чтение и запись в файл	21 833	29 719

В процессе выполнения тестов по запуску программ с использованием контейнеризации и без использования, было выяснено, что приложение, выполняющее исключительно арифметические операции без контейнера, выполнилось не существенно быстрее, на 1,7 %, а программа, выполняющая чтение и запись в файл без контейнера, выполнялась на 26,5 % быстрее. Такое значительное замедление программы при использовании контейнеризации следует учитывать перед принятием решения о ее применении.

Список литературы

1. Милл И., Сейерс Э. Х. Docker на практике / пер. с англ. Д. А. Беликов. М. : ДМК Пресс, 2020. 516 с.
2. Официальный сайт проекта Docker. URL: https://www.docker.com (дата обращения: 14.10.2024).
3. Шилдт Г. Java 8. Полное руководство. М. : Вильямс, 2015. 1376 с.

Информация об авторах

Холодков Данила Валерьевич, аспирант, Пензенский государственный университет.

Зинкин Сергей Александрович, доктор технических наук, профессор, профессор кафедры «Вычислительная техника», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004.414.2:004.085.3

СРАВНЕНИЕ ВРЕМЕНИ, НЕОБХОДИМОГО ДЛЯ ЗАПИСИ ОПТИЧЕСКИХ ДИСКОВ, ОРГАНИЗОВАННЫХ В СТРУКТУРЫ R1 И G16

А. В. Чернышов

Мытищинский филиал Московского государственного
технического университета им. Н. Э. Баумана,
Мытищи, Россия

sch-ru@yandex.ru

Аннотация. Выполнено сравнение времени, необходимого на запись одного и того же по объему блока информации на оптические диски однократной записи, объединенные в две разные структуры: одиночные диски с несколькими копиями (структура R1) и гибридная структура на базе RAID16 (структура G16). Сравнение выполнено для сопоставимых вероятностей потери информации. Показано, что при небольшом количестве копий структура G16, выигрывая у структуры R1 по количеству оптических дисков, проигрывает по времени записи. Однако с увеличением количества копий при сохранении выигрыша G16 по количеству оптических дисков время записи блока информации практически сравнивается.

Ключевые слова: архивы долговременного хранения электронной информации, оптические диски однократной записи, массивы RAID оптических дисков, объединение оптических дисков в гибридные структуры

Для цитирования: Чернышов А. В. Сравнение времени, необходимого для записи оптических дисков, организованных в структуры R1 и G16 // Вестник Пензенского государственного университета. 2024. № 4. С. 137–139.

Введение

При создании файловых хранилищ долговременного хранения архивной информации на базе оптических дисков однократной записи[1] для минимизации вероятности потери информации в процессе хранения из-за неизбежной деградации дисков [1] эти диски могут быть организованы в различные структуры. Если просто создаются группы из n одиночных оптических дисков с идентичными копиями информации [2, 3], то фактически образуется зеркальная структура типа RAID1 (обозначим ее как R1). Недостатком этой структуры является необходимость создания довольно большого количества копий n для достижения требуемой минимальной вероятности потери информации. Существенно уменьшить количество необходимых копий и, как следствие, необходимое количество оптических дисков позволяет объединение дисков в гибридные структуры на базе массивов RAID16 (обозначаемый далее как G16). Любой массив RAID можно условно рассматривать как группу k дисков с информацией и группу m дисков с контрольными суммами. Для массива RAID16 всегда $m = 2$. Значение k должно быть не меньше 2. И именно при $k = 2$ массив RAID16 имеет минимальную вероятность потери информации из-за выхода из строя, поэтому именно в такой конфигурации его лучше всего применять для длительного хранения информации.

[1] ГОСТ Р 54989–2012 / ISO TR 18492:2005 Обеспечение долговременной сохранности электронных документов (вступ. в силу 01.05.2013).

Соответственно для структур G16 в одном блоке при $k = 2$ имеется два одиночных диска с разной информацией и $n–1$ массив дисков RAID16, на которые записана та же самая информация, что и на два одиночных диска из этого блока. Зная время t_w, необходимое для записи одного оптического диска, несложно вычислить суммарное время, необходимое для записи одного блока информации структуры R1. Однако для структуры G16 в процессе записи блока информации необходимо выполнение дополнительных операций с записываемой информацией на жестком диске [4], что предполагает изменение суммарного времени, необходимого для записи одного блока информации, по отношению к такому же количеству оптических дисков в структуре R1. Это приводит к изменению значения параметра t_w, которое необходимо учитывать для корректной оценки затрат времени в расчетах для гибридных структур. Однако в связи с тем, что при сопоставимых вероятностях потери информации количество оптических дисков в блоке структуры G16 существенно меньше, чем в блоке R1 такой же информационной емкости, характер этого изменения значения t_w неочевиден, и его необходимо оценить.

Целью работы является сравнительная оценка значений параметра t_w (время записи одного оптического диска) для нескольких значений n (количества запасных копий информации) для структур оптических дисков R1 и G16.

Решение задачи

Рассмотрим тенденцию изменения времени записи одного блока информации для нескольких значений параметра n структур G16. При этом будем рассматривать структуру G16 с параметрами: $k = 2$, $m = 2$. Структура G16 при $n = 3$ имеет то же количество оптических дисков (10), что и структура R1 при $n = 5$. Но при этих параметрах структура G16 имеет значительно меньшую вероятность потери информации при хранении (конкретные значения вероятностей в данном случае несущественны, поскольку зависят от объема архива и типа применяемых оптических дисков). Расчеты показывают, что для структуры G16 при $n = 3$ вероятность потери информации будет меньше, чем для структуры R1 при $n = 6$, и больше, чем для структуры R1 при $n = 7$, а если рассмотреть структуру G16 при $n = 4$, то вероятность потери информации будет меньше, чем для структуры R1 при $n = 9$, и больше, чем для структуры R1 при $n = 10$.

При оценке затрат времени будем использовать экмпериментально полученное для структуры R1 среднее значение времени записи одного оптического диска с верификацией t_w, составляющее 33 мин 40 с. Для удобства расчетов запишем его в секундах: $t_w = 2020$ с. Проще всего оценить суммарное время, необходимое для записи блока информации для структуры R1 (табл. 1).

Таблица 1

Оценка суммарного времени, необходимого для записи блока информации для структуры R1

Количество копий информации n	5	6	7	9	10
Количество дисков в блоке информации, шт.	10	12	14	18	20
Общее время записи блока информации, с	20 200	24 240	28 280	36 360	40 400

Для оценки времени записи одного гибридного блока G16 необходимо оценить время, затрачиваемое на каждую операцию (табл. 2). Все длительности времени в данной таблице приведены в секундах. Порядок операций изменен (сжат) для удобства обобщения.

Как можно видеть, процесс записи одного блока информации на оптические диски, организованные в структуры G16 при $n = 3$, требует в 1,57 раза больше времени, чем для структур R1 для $n = 5$, в 1,31 раза для $n = 6$ и в 1,11 раза для $n = 7$. Однако количество затрачиваемых при этом оптических дисков на запись блока информации существенно меньше, что позволяет экономить затраты на закупку оптических дисков, обеспечивая при этом сопоставимую невысокую вероятность потери информации при хранении. Для структуры G16 при $n = 4$ превышение времени по отношению к сопоставимой по вероятности потери информации структуре R1 для $n = 9$ составля-

ет 1,09 раза и для n = 10 – 0,98 раза, т.е. в последнем случае обеспечивается выигрыш не только по количеству оптических дисков, но и по времени записи.

Таблица 2

Оценка суммарного времени, необходимого для записи блока информации для структуры G16

Операции	n = 3 (10 дисков)	n = 4 (14 дисков)
Запись одиночных дисков		
Формирование образов d1.iso, d2.iso	540×2	540×2
Контроль целостности образов d1.iso, d2.iso	196×2	196×2
Запись образов d1.iso, d2.iso на оптические диски	2020×2	2020×2
Формирование массива RAID-6		
Сжатие образов d1.iso, d2.iso	3000×2	3000×2
Контрольное разжатие образов d1.iso, d2.iso	660×2	660×2
Контроль целостности разжатых образов d1.iso, d2.iso	540×2	540×2
Запись образов дисков массива RAID-6 на оптические диски		
Формирование контрольных сумм образов дисков	196×4	196×4
Контроль полученных контрольных сумм	196×4	196×4
Запись оптического диска массива RAID-6 № 1	2020×2	2020×3
Запись оптического диска массива RAID-6 № 2	2020×2	2020×3
Запись оптического диска массива RAID-6 № 3	2020×2	2020×3
Запись оптического диска массива RAID-6 № 4	2020×2	2020×3
Итого	31 640	39 720
Оценка t_w	3164 с (52 мин 44 с)	2837 с (47 мин 17 с)

Заключение

Для структуры оптических дисков G16 при n = 3 значение параметра t_w существенно отличается в большую сторону от значения t_w для структуры R1 при сопоставимых вероятностях потери информации, и увеличение суммарного времени записи блока информации для структуры G16 должно учитываться при проведении расчетов параметров архивных хранилищ информации. В то же время для структуры G16 при n = 4 значение параметра t_w хотя и превышает значение t_w для структуры R1, но суммарное время записи блока информации для структур G16 и R1 отличается незначительно.

Список литературы

1. Пилипчук М. И., Балакирев А. Н., Дмитриева Л. В., Залаев Г. З. Рекомендации по обеспечению сохранности информации, записанной на оптических дисках (тестирование выборочного массива документов федеральных архивов). М. : РГАНТД, 2011. 52 с.

2. Zheng J., Slattery O. T. NIST/Library of Congress Optical Disc Longevity Study: Final Report. September, 2007. 32 p. URL: https://www.loc.gov (дата обращения: 02.10.2024).

3. Рекомендации по комплектованию, учету и организации хранения электронных архивных документов в государственных и муниципальных архивах. М. : Федеральное архивное агентство. ВНИИДАД, 2013. 49 с.

4. Чернышов А. В. Метод размещения информации в долговременном электронном архиве, выполненном на оптических дисках однократной записи, организованных в гибридные структуры // Новые информационные технологии и системы : сб. науч. ст. по материалам XVII Междунар. науч.-техн. конф. (г. Пенза, 18–19 ноября 2020 г.). Пенза : Изд-во ПГУ, 2020. С. 8–12.

Информация об авторе

Чернышов Александр Викторович, кандидат технических наук, доцент, доцент кафедры «Прикладная математика и вычислительная техника», Мытищинский филиал Московского государственного технического университета им. Н. Э. Баумана.

Автор заявляет об отсутствии конфликта интересов.

УДК 004.024

ОБЗОР АДАПТИВНЫХ МЕХАНИК ИГРЫ

А. В. Чистоходов [1], В. В. Эпп [2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1] achistohodov@gmail.com
[2] vitalinae@mail.ru

Аннотация. Адаптивные механики игры представляют собой изменения, происходящие во время игрового сеанса по желанию игрока или согласно внутренним алгоритмам самой игры. Разработка процесса такой механики направлена на подстройку игрового контента под предпочтения, навыки и стиль игры пользователя. Эта механика позволит увеличить вовлеченность различных категорий игроков и может улучшить восприятие игрового процесса.

Ключевые слова: игра, адаптивная механика, алгоритмы адаптации, игровая механика

Для цитирования: Чистоходов А. В., Эпп В. В. Обзор адаптивных механик игры // Вестник Пензенского государственного университета. 2024. № 4. С. 140–142.

Игровая индустрия стремительно развивается, предлагая игрокам все больше разнообразных игр и разновидностей игровых механик. Одной из последних тенденций было внедрение адаптивных механик, которые позволяют игре подстраиваться под конкретного игрока, делая процесс прохождения более увлекательным и индивидуальным для каждого [1].

Что такое адаптивные механики?

Адаптивные механики – это элементы игры, которые изменяются в зависимости от действий, предпочтений или уровня мастерства игрока. Это может касаться сложности уровней, скорости реакции врагов, изменения условий победы или даже настройки всего уровня. Основная цель таких механик – сделать игру максимально комфортной для каждого отдельного игрока, независимо от его опыта или стиля игры.

Типы игровых механик.

Прежде чем углубиться в особенности адаптивных решений, важно понять, какие вообще существуют игровые механики [2, 3]:

– *механики управления*. То, каким образом игрок взаимодействует с игрой. Например, управление персонажем через клавиатуру или джойстик;

– *боевые механики*. Эти механики касаются боя и взаимодействия с врагами. Сюда входят системы атак, уклонения и получение урона;

– *экономические механики*. Включают в себя сбор ресурсов, торговлю, улучшение персонажей и зданий, развитие экономики внутри игры;

– *прогрессивные механики*. Связаны с развитием персонажа или прогресса в игре. Например, «прокачка» умений, открытие новых локаций, получение новых предметов;

– *социальные механики*. Механизмы взаимодействия между игроками, включая кооперативный режим, многопользовательские сражения, обмен ресурсами и общение как сообщениями, так и голосом;

– *сюжетные механики*. Элементы, связанные с сюжетом игры, такие как принятие решений, влияние выбора на дальнейшее развитие событий, интерактивные диалоги.

Адаптивные механики в играх.

Адаптивные механики – это те же стандартные механики, но с учетом того, что они подстраиваются персонально. Существует множество их разновидностей, некоторые из них представлены на рис. 1.

1. Динамическая сложность.

Одна из самых популярных форм адаптации – изменение уровня сложности в зависимости от успехов игрока [4]. Если игра замечает, что игрок часто умирает на одном уровне, она может снизить количество врагов или уменьшить урон, наносимый ими. В случае успешного прохождения уровней без особых трудностей, наоборот, сложность увеличивается. Примером такой механики является серия игр Dark Souls, Resident Evil, где сложность изменяется динамически в зависимости от того, насколько хорошо вы играете.

2. Персонализация контента.

Современные игры все чаще предлагают игроку возможность выбирать, каким путем он пойдет, станет ли он охотником, охотившимся за дичью, торговцем, который перепродает товары, или воином, не знавшим жалости. Так, в ролевых играх (RPG) игроки могут выбрать путь развития своего персонажа, который будет влиять на задания (квесты), предметы и сюжетные линии. Такие игры, как The Witcher 3 или Skyrim, предоставляют множество вариантов для персонализации, позволяя каждому игроку пройти игру по-своему.

Рис.1. Виды адаптивных механик

3. Интеллектуальные противники.

Противники в играх оживают благодаря использованию искусственного интеллекта (AI). Современные алгоритмы позволяют врагам анализировать поведение игрока и изменять свое поведение в реальном времени. Это делает бои более интересными и непредсказуемыми, так как каждый раз игрок сталкивается с другим противником. Пример такого алгоритма представлен в линейки игр XCOM. Враги в этой игре обладают продвинутым AI, который позволяет им анализировать действия игрока и реагировать соответствующим образом. Это делает каждую битву уникальной и требует от игрока постоянного анализа ситуации.

4. Адаптация интерфейса.

Адаптивным может быть и интерфейс. Так, например, если пользователь хочет играть на джойстике или тачпаде, а не на клавиатуре с мышкой, игра может автоматически изменить интерфейс расположения кнопок управления. Некоторые игры изменяют интерфейс в зависимости от разрешения экрана.

5. Аудиовизуальная адаптация.

Звуковые эффекты и визуальное оформление могут меняться в зависимости от сцены в игре, например, музыка становится более напряженной, когда наступает опасность. Графика тоже может меняться для акцентирования на определенных моментах сюжета или для того, чтобы усилить реакцию игрока.

Преимущества использования адаптивных механик представлены на рис. 2.

Рис. 2. Преимущества использования адаптивных механик

Адаптивная механика – это неотъемлемая часть современной игровой индустрии. Она делает игры более увлекательными и индивидуальными. Развитие технологий и искусственного интеллекта открывает новые возможности для создания более сложных и интересных игровых систем.

Список литературы

1. Алимов А. А., Давтян Ф. Г., Катаев А. В., Шабалина О. А. Адаптивные обучающие игры как тренд развития обучающего ПО // Информационные технологии в науке, образовании и управлении. 2018. № 4. С. 11–15.

2. Основные типы игровых механик. URL: https://sky.pro (дата обращения: 29.10.2024).

3. Что такое игровая механика? URL: https://skillbox.ru (дата обращения: 29.10.2024).

4. Динамическая сложность в играх. URL: https://sky.pro (дата обращения: 29.10.2024).

Информация об авторах

Чистоходов Артемий Владимирович, студент, Пензенский государственный университет.

Эпп Виталина Викторовна, кандидат технических наук, доцент, доцент кафедры «Системы автоматизированного проектирования», Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 004:042

ОБЗОР МОДЕЛЕЙ ПРЕДСТАВЛЕНИЯ АКТИВНЫХ ПРАВИЛ В ДИСКРЕТНО-СОБЫТИЙНЫХ ИНФОРМАЦИОННО-УПРАВЛЯЮЩИХ СИСТЕМАХ

С. В. Шибанов[1], Я. С. Шлепнев[2]

[1, 2]Пензенский государственный университет, Пенза, Россия

[1] sergey.v.shibanov@yandex.ru
[2] yaroslav.shlepnev@yandex.ru

Аннотация. Проведен обзор формальных и графических моделей для описания событийных процессов и выполнения активных правил в дискретно-событийных реагирующих информационно-управляющих системах. Рассматриваются различные подходы к представлению событий (включая использование условных и временных операторов, которые позволяют более точно управлять активацией и последовательностью выполнения правил). Особое внимание уделено операциям над событиями с учетом временной семантики их возникновения, что важно для правильной интерпретации сложных событий и их комбинаций. Также обсуждаются методы повышения эффективности алгоритмов обнаружения и устранения проблемных ситуаций при исполнении активных правил, таких как конфликты, зацикливания и другие ошибки, возникающие при параллельной или последовательной активации событий.

Ключевые слова: активные правила, формальные модели, информационно-управляющие системы, графы отображения событий, сети Петри, алгебра логики

Для цитирования: Шибанов С. В., Шлепнев Я. С. Обзор моделей представления активных правил в дискретно-событийных информационно-управляющих системах // Вестник Пензенского государственного университета. 2024. № 4. С. 143–148.

Введение

Современные информационно-управляющие системы (ИУС) играют ключевую роль в различных отраслях промышленности, транспорта, энергетики и других секторах экономики. Эффективное управление этими системами требует оперативной обработки и анализа большого объема данных, поступающих в реальном времени. Важным аспектом является разработка алгоритмов, которые способны анализировать потоки событий и на их основе формировать управляющие воздействия, обеспечивая тем самым высокую адаптивность и устойчивость ИУС.

Правила типа «событие – условие – действие» (ECA) [1] обладают достаточно высокой выразительностью для описания сложных событий и реакций на них. Поэтому этот событийно-управляемый формализм широко используется для спецификации сложных информационно-управляющих систем [2], например, для управления в промышленных масштабах и повышения эффективности, особенно при интеграции с технологиями, такими как встроенные системы и сети датчиков.

Правила ECA используются для задания реакции системы на события и записываются в формате: «при наступлении набора событий, если выполняются определенные условия, выпол-

нить данные действия». Однако для систем с большим количеством компонентов и сложным поведением может быть трудно корректно задать такие правила [3].

Классический подход к описанию активных правил – это использование нотации «событие – условие – действие» (Event – Condition – Action, ECA) [4]. В этом подходе выделяются типы событий и их экземпляры. Все события делятся на группы по типам, каждому из которых присваиваются имя и параметры с указанными типами данных, выступающие в качестве входных данных для активного правила. Конкретное событие определенного типа, которое происходит в системе, называют экземпляром события; оно включает информацию о типе, времени возникновения и контексте, представляющем собой набор значений параметров.

Условие – это логическое выражение, основанное на контексте события, определяющее, должна ли система реагировать на событие или проигнорировать его.

Действие, прописанное в активном правиле, представляет собой цепочку управляющих операций, выполняемых как отклик на событие и направленных на управляемый объект ИУС. Параметры этих операций становятся выходными параметрами правила [5].

Проблемы исполнения активных правил

Встраивание активных правил (ECA) в дискретно-событийные реагирующие динамические системы реального времени направлено на оптимизацию процесса управления с помощью четко определенного набора правил. Активные правила применяются в информационно-управляющих системах реального времени [6], системах поиска информации и интеллектуального анализа текста, а также в сценариях мониторинга статистических изменений в веб-приложениях [7]. Тем не менее, в ходе выполнения активных правил возможны конфликты, связанные с их взаимодействием.

Конфликты активных правил считаются основной угрозой в процессе обнаружения и обработки событий.

Так, одной из проблемных ситуаций активных правил является несогласованность правил: два правила считаются несогласованными, если порядок их выполнения может привести к разным результатам; т.е. выполнение e1, а затем e2 приводит к иному результату, чем выполнение e2, а затем e1. Смысловая несогласованность действий, выполняющихся в результате активации правил, может привести рассматриваемые объекты управления к некорректному состоянию и нарушить целостность информационно-управляющей системы.

Действие активного правила может являться событием другого правила, что приводит к последовательной активации целого набора правил. Без наличия специальных алгоритмов контроля этот процесс может продолжаться бесконечно, что приводит к проблеме незавершенности правил или ситуации зацикливания [8].

В процессе взаимодействия активных правил может возникать состояние гонки (race condition) – ситуация, когда последовательная активация различных правил приводит к изменению состояния одного и того же объекта управления, при этом активация второго правила отменяет результат активации первого правила.

Одной из главных особенностей дискретных реагирующих систем являются наличие быстрой реакции на возникновения событий и их обработка в режиме реального времени. Тем не менее, возникновение описанных проблемных ситуаций может приводить к деградации пропускной способности и к нарушению целостности системы (дискретные системы в тот или иной момент времени должны находиться в определенном состоянии). В статье проводится обзор формальных и графических моделей и приводится набор операций над событиями с учетом временной семантики их возникновения с целью дальнейшего построения эффективных алгоритмов выявления и устранения проблемных ситуаций исполнения активных правил.

Обзор формальных моделей представления активных правил

Данный раздел посвящен проблеме представления и формализации правил, а также приводятся решения и попытки их реализации.

В настоящее время было предпринято множество попыток представить правила как в формальной, так и в графической форме. Формальное представление можно найти в языках запросов, таких как в реляционных и XML-базах данных [9].

В работе [5] для моделирования процесса взаимодействия активных правил используют нотацию взаимодействующих последовательных процессов (CSP, Communicating Sequential Processes), разработанную Ч. Хоаром. Взаимодействующие последовательные процессы (CSP) – формальный язык для описания моделей взаимодействия в параллельных системах, который относится к математическим теориям параллелизма, известным как исчисление процессов (или алгебра процессов).

Авторы проекта HiPAC [9], изучающие активные базы данных, разработали формальную модель, которая считается эталоном вербального представления правил для различных систем, расширяющую базовую нотацию ECA с добавлением времени исполнения, режима исполнения и пользовательских атрибутов правила.

Данные формальные представления применяются в различных информационно-управляющих системах, но имеют существенные недостатки относительно графических, не только с точки зрения полноты описания правил, но и с точки зрения конструирования правил и их взаимодействия. Описанные формальные модели учитывают критические аспекты взаимодействия правил, в ходе которых могут происходить различные проблемные ситуации (коллизии), но не позволяют конструировать алгоритмы, позволяющие выявить и устранить потенциальные угрозы процесса исполнения.

Обзор систем и моделей графического представления активных правил

Объектно-ориентированные базы данных (OODB) с подходом отображения, основанным на применении правил, представляют собой системы, в которых правила организуются и визуализируются в виде слоев. Это позволяет структурировать правила, улучшить их наглядность и управляемость, особенно в контексте сложных данных. Такой подход используется в различных моделях, например, **SAMOS-PN** в системе SAMOS, а также в моделях **Snoop** и **SnoopIB**, где графическое представление слоя правил способствует лучшему пониманию логики и структуры данных [10, 11].

Система **SnoopIB** (Snoop Interval-Based) позволяет моделировать процесс исполнения активных правил с использованием графа обнаружения событий (Event Detection Graph, EDG), в котором простые события отображаются в виде листовых узлов, а внутренние узлы представляют составные события. Система **SnoopIP** является развитием системы **Snoop**, которая, в свою очередь, основывается на **Sentinel**. Одной из особенностей данной системы является использованием операторов булевой алгебры для представления различных классов событий с учетом их временной семантики возникновения.

Так, оператор ЛЮБОЙ может представлять операцию конъюнкции, оператор АПЕРИОД позволяет представить события, которые могут происходить более одного раза в течение заданного интервала времени, а оператор ПЕРИОД работает аналогично АПЕРИОД, но с фиксированным интервалом активации события в пределах заданного интервала времени.

Тем не менее, граф обнаружения событий обладает следующими существенными недостатками:

– возможное создание дубликатов одних и тех же событий, что приводит к избыточности и увеличению объема памяти;

– недостаточно гибкие механизмы для учета условий активации правил, что ограничивает сложные сценарии исполнения;

– меньшая эффективность в обработке комбинаций событий по сравнению с более специализированными системами;

– отсутствие возможности выявления и устранения критических ситуаций исполнения.

Все это привело к необходимости возникновения более гибкой и адаптивной модели отображения.

Система **SAMOS** (Swiss Active Mechanism-based Object-oriented database System) – это объектно-ориентированная база данных с применением механизма активных правил. Эта система реализует собственную модель графического отображения активных правил, которая называется SAMOS-PN. Данная система основывается на классических сетях Петри, но имеет критический недостаток в виде отсутствия условных переходов для управления сложными процессами управления.

CCPN (Conditional Colored Petri Nets) – это разновидность цветных сетей Петри (Colored Petri Nets, CPN), которая добавляет условные переходы для управления сложными процессами. В классической цветной сети Петри каждый маркер (токен) может иметь цвет или атрибут, который помогает различать данные, проходящие через сеть. Однако в условной цветной сети Петри добавляется еще один уровень контроля – условия, которые позволяют гибко управлять переходами. Эти условия могут зависеть от значений токенов или от внешних параметров, что позволяет активировать или блокировать переходы в зависимости от определенных состояний.

По сравнению с графами обнаружения события и классическими сетями Петри условно раскрашенные сети Петри имеют следующие преимущества:

– отсутствие дублирования событий и контекстов событий;

– учитывается условие активации при обработке правил, что обеспечивает более точное управление активными правилами и адаптацию к различным контекстам и сценариям исполнения;

– обеспечиваются более эффективное обнаружение и обработка составных (сложных) событий, что делает его подходящим для сложных сценариев, где события могут взаимодействовать друг с другом.

Сравнительный анализ систем и моделей представления активных правил и наборов операторов алгебры логики

Описанные программные средства представляют активные правила различными способами. Системы Sentinel, Snoop, SnoopIB в качестве модели отображения используют графы обнаружения событий, в то время как система SAMOS основывается на применении классической сети Петри.

Для моделирования составных событий в указанных системах используются логические операции алгебры логики. Активные правила могут рассматриваться как предикаты, поскольку они определяют условия (предикаты), которые должны быть истинными для активации действия. Активные правила могут быть представлены в рамках исчисления предикатов, где условия и действия могут быть представлены как логические выражения. Например, активное правило может быть описано в виде формулы, включающей логические операторы, такие как AND, OR и NOT, и кванторы, такие как «для всех» (∀) или «существует» (∃).

С точки зрения операторов алгебры логики можно выделить следующие основные операции над событиями:

– конъюнкция событий (И): И (e_1, e_2, …, e_n);

– дизъюнкция событий (ИЛИ): ИЛИ (e_1, e_2, …, e);

– отрицание события (НЕ) (по интервалу): НЕ (e_1) в интервале [t_1, t_2];

– последовательность событий (ПОСЛЕД): ПОСЛЕД (e_1, e_2, …, e_n);

– любое событие (ЛЮБОЕ): ЛЮБОЕ (m, e_1, e_2, …, e_n), где $m \le n$, m – количество активированных событий, n –количество всех событий;

– замыкание события (ПЕРВ): ПЕРВ (e_1) в интервале [t_1, t_2], т.е. первое появление события e_1.

Описанные средства представления активных правил в той или иной степени реализуют основные логические операции над событиями. Тем не менее, с целью построения логики исполнения активных правил с учетом критических аспектов их взаимодействия и расширенных классов событий авторы рассмотренных систем вводят новые логические операции для учета временной семантики возникновения событий.

Система Snoop, основанная на графах отображения событий, расширяет набор логических операторов для представления периодических (с фиксированным интервалом возникновения) и апериодических (без фиксированного интервала возникновения) событий.

Система SnoopIB предоставляет дополнительные операторы для поддержки обнаружения составных (сложных) событий, такие как операторы AND, NOT и PLUS, которые не были представлены в Snoop. Такое дополнение графа отображения событий является большим преимуществом по сравнению с моделью раскрашенной сети Петри (S-PN), применяемой в систему SAMOS. Тем не менее, наличие дубликатов в графах обнаружения событий все еще является существенным недостатком по сравнению с моделью CCPN, в которой отсутствует дублирование событий и состояний.

Классические сети Петри, применяемые в SAMOS (S-PN), используются для представления потоков входящих событий их параметров, в которой дочерние выражения представлены отдельно, что может привести к их дублированию в модели (является также большим недостатком системы и раскрашенных сетей Петри).

Модель условно раскрашенной сети Петри, в свою очередь, поддерживает все рассмотренные логические операторы, расширяя набор операторов логической функцией ОДНОВРЕМЕННО, которая отслеживает возникновение набора событий в течение заданного промежутка времени.

В табл. 1 представлен сравнительный анализ наличия логических операторов в описанных системах.

Таблица 1

Сравнительный анализ наличия логических операторов в системах и моделях представления активных правил

Оператор алгебры логики	Sentinel (EDG)	Snoop (EDG)	SnoopIB (EDG)	SAMOS (PN)	CCPN
И	✓	–	✓	✓	✓
ИЛИ	✓	✓	✓	✓	✓
ЛЮБОЕ	✓	✓	✓	–	✓
ПЕРВ	–			✓	✓
НЕ	–	–	✓	✓	✓
ПОСЛЕД	–	✓	✓	✓	✓
ПЕРИОД	–	✓	✓	–	✓
АПЕРИОД	–	✓	✓	–	✓
ОДНОВРЕМЕННО	–	–	–	–	✓

Так, набор всех операторов с учетом временной семантики возникновения событий для описания сложных событий с использованием условно раскрашенной сети Петри может выглядеть следующим образом:

– конъюнкция событий (И): И (e_1, e_2, …, e_n);

– дизъюнкция событий (ИЛИ): ИЛИ (e_1, e_2, …, e);

– отрицание события (НЕ) (по интервалу): НЕ (e_1) в интервале [t_1, t_2];

– последовательность событий (ПОСЛЕД): ПОСЛЕД (e_1, e_2, …, e_n);

– любое событие (ЛЮБОЕ): ЛЮБОЕ (m, e_1, e_2, …, e_n), где $m \leq n$, m – количество активированных событий, n – количество всех событий;

– замыкание события (ПЕРВ): ПЕРВ (e_1) в [t_1, t_2], т.е. первое появление события e_1;

– периодическое возникновение: ПЕРИОД (e_1, [t_1, t_2], t_3), где [t_1, t_2] – временной интервал, t_3 – частота возникновения события;

– апериодическое возникновение: АПЕРИОД (e_1, [t_1, t_2]), где [t_1, t_2] – временной интервал;

– одновременное возникновение событий: ОДНОВРЕМЕННО (e_1, e_2, …, e_n, [t_1, t_2]), где [t_1, t_2] – временной интервал.

Заключение

Таким образом, предложенное формализованное и математически обоснованное представление событий в виде условной раскрашенной сети Петри с применением логических операторов, учитывающих временную семантику возникновения событий, позволяет выстраивать эффективные алгоритмы выявления и устранения коллизий исполнения активных правил, что приводит к повышению эффективности управления дискретно-событийных информационно-управляющих систем.

Список литературы

1. McCarthy D., Dayal U. The architecture of an active database management system // ACM SIGMOD Record. 1989. № 18 (2). P. 215–224.

2. Abadi D. J., Carney D., Çetintemel U. [et al.]. Aurora: a new model and architecture for data stream management // The VLDB Journal. 2003. № 12 (2). P. 120–139.

3. Augusto J. C., Nugent C. D. A new architecture for smart homes based on ADB and temporal reasoning // Toward a Human-Friendly Assistive Environment. 2004. Vol. 14. P. 106–113.

4. Caroprese L., Truszczyński M. Declarative Semantics for Active Integrity Constraints // Logic Programming. Lecture Notes in Computer Science. 2008. № 5366. P. 269–283.

5. Шибанов С. В., Скоробогатько А. А., Лысенко Э. В. Интегрированная модель активных правил // Математическое и программное обеспечение систем в промышленной и социальной сферах. 2011. № 1-1. С. 41–46.

6. Шибанов С. В., Шлепнев Я. С. Сервис потоковой обработки событий и исполнения активных правил // Математическое моделирование и суперкомпьютерные технологии : тр. XXI Междунар. конф. (г. Н. Новгород, 22–26 ноября 2021 г.). Н. Новгород : Национальный исслед. Нижегород. гос. ун-т им. Н. И. Лобачевского, 2021. С. 403–407.

7. Zimmer D., Meckenstock A., Unland R. Using Petri Nets for Rule Termination Analysis // Proceedings of the workshop on on Databases: active and real-time. 1997. P. 29–32. URL: researchgate.net>

8. Couchot A. Improving Termination Analysis of Active Rules with Priorities // DEXA. 2003. Vol. 3. P. 846–855.

9. Dayal U., Blaustein B., Buchmann A. P. [et al.]. The HiPAC Project: Combining Active Databases and Timing Constraints // SIGMOD Record. 1988. Vol. 17. P. 51–70.

10. Gatziu S. Dittrich K. R. SAMOS: An active object-oriented database system // IEEE Quartely Bulletin on Data Engineering. 1993. Vol. 15. P. 23–26.

11. Dittrich K. R., Fritschi H., Gatziu S. [et al.]. SAMOS in hindsight: experiences in building an active object-oriented DBMS // Information Systems. 2003. Vol. 28. P. 369–392.

Информация об авторах

Шибанов Сергей Владимирович, кандидат технических наук, доцент, доцент кафедры «Математическое обеспечение и применение электронных вычислительных машин», Пензенский государственный университет.

Шлепнев Ярослав Сергеевич, аспирант, Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 538.9

ЭФФЕКТЫ 2D-ДИССИПАТИВНОГО ТУННЕЛИРОВАНИЯ В ПРЕДЕЛЕ СИЛЬНОЙ ДИССИПАЦИИ

В. Д. Кревчик[1], М. Б. Семёнов[2], П. В. Кревчик[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1, 2, 3]physics@pnzgu.ru

Аннотация. Рассмотрена модель 2D-диссипативного туннелирования в условиях внешнего электрического поля при конечной температуре с учетом влияния двух локальных фононных мод матрицы – термостата. Показано, что в определенном диапазоне экспериментально реализуемых параметров наблюдается режим 2D-бифуркаций и квантовых биений. Предсказан этот 2D-эффект на полевой зависимости интенсивности фотолюминесценции на примере двуслойных структур туннельно-связанных полупроводниковых квантовых точек InAs/GaAs (001).

Ключевые слова: диссипативное туннелирование, локальные фононные моды, электрическое поле, режим бифуркаций и квантовых биений, квантовые точки

Для цитирования: Кревчик В. Д., Семёнов М. Б., Кревчик П. В. Эффекты 2D-диссипативного туннелирования в пределе сильной диссипации // Вестник Пензенского государственного университета. 2024. № 4. С. 150–159.

В работе «Будущие полупроводниковые устройства для проектирования многозначных логических схем» авторов Supriya Karmakar, Faquir C. Jain [1] представлены будущие устройства для реализации многозначной логики. Полевой транзистор с квантовыми точками (Quantum dot gate field effect transistor, QDGFET) работает на основе изменения порогового напряжения из-за накопленного заряда в квантовых точках в области затвора. Полевой транзистор с каналом квантовых точек (Quantum dot channel field effect transistor, QDCFET) создает большее количество состояний в своих передаточных характеристиках из-за потока заряда через структуру мини-зоны, образованной перекрывающимися энергетическими зонами соседних квантовых точек в области канала полевого транзистора. С другой стороны, полевой транзистор с переключением пространственной волновой функции (spatial wave-function switched field effect transistor, SWSFET) создает большее количество состояний в своей передаточной характеристике на основе переключения носителей заряда с одного канала на другой канал устройства. В этой статье [1] авторы подробно обсуждают QDGFET, QDCFET и SWSFET транзисторы, чтобы изучить их применение в будущих многозначных логических схемах. *Впервые в исследуемых приборах используется туннельный транспорт в двойных слоях квантовых точек.*

В работе авторов Eliade Stefanescu, Werner Scheid «Сверхизлучательное диссипативное туннелирование в двойной *p*-*i*-*n*-полупроводниковой гетероструктуре с термической инжекцией электронов» [2] предлагается полупроводниковый прибор с двумя *p*-*i*-*n*-переходами, поддерживаемыми при двух различных температурах. Когда ток, подаваемый в устройство из-за этой разницы температур, превышает пороговое значение, создается «сверхизлучательное» поле в первом затворе, которое индуцирует дополнительный ток во втором затворе. Этот реакционный контур

усиливает инжекционный ток. Таким образом, тепловой поток между двумя контактами частично преобразуется в «сверхизлучательную» мощность. Предсказанное Дике [2] «сверхизлучение» интенсивно изучалось с учетом различных физических эффектов, таких как: 1) статистическое распределение электронных состояний, 2) эффекты вырождения уровней, 3) силы Ланжевена, действующие на атомную систему, 4) поперечные эффекты, 5) конкуренция однофотонных и трехфотонных переходов, 6) спектр «сверхизлучения», 7) наличие фотонных щелей, 8) спонтанно генерируемые эффекты когерентности, 9) подавление «сверхизлучения» за счет рассеяния [2]. Однако новый особый интерес к «сверхизлучению» сейчас возникает из-за возможных приложений в информационных технологиях, с одной стороны, и, с другой стороны, из-за возникающих очень сложных вопросов в задаче диссипативного взаимодействия атома с полем [2]. В принципе, квантовая диссипация (квантовое туннелирование с диссипацией) – сложное явление, описание которого зависит от модели, принятой для диссипативной связи, и от процедуры, используемой для сведения гамильтонова уравнения системы и окружающей среды к квантовому кинетическому уравнению (a quantummasterequation) [2].

Авторы (Nan Maand, Debdeep Jena) статьи «Межзонное туннелирование в двумерных кристаллах полупроводников» [3] исследуют межзонное квантовое туннелирование электронов в полупроводниках, которое в последнее время вызывает повышенный интерес как основной механизм переноса в туннельных полевых транзисторах. Такие транзисторы потенциально могут выполнять электронное переключение с меньшей энергией, чем их обычные аналоги. Недавнее появление двумерных (2D) полупроводниковых кристаллов обеспечивает привлекательную материальную основу для реализации таких устройств. В работе [3] авторы выводят аналитическое выражение для понимания потока туннельного тока в однослойных двумерных кристаллических полупроводниках в *k*-пространстве. Авторы применяют полученные результаты к ряду двумерных кристаллических полупроводников и сравнивают их с туннельными токами в трехмерных полупроводниках. Также обсуждается применимость полученных результатов для туннельных устройств.

В работе авторов H. S. Borges, L. Sanz, A. M. Alcalde «Диссипативная динамика в связанных квантовых точках: управление туннелированием и электромагнитно-индуцированная прозрачность» [4] исследована диссипативная динамика двух асимметричных туннельно-связанных квантовых точек в присутствии лазерного когерентного излучения. Решая уравнение Лиувилля – фон Неймана – Линдблада, связанное с трехуровневым модельным гамильтонианом, авторы анализируют эффект спонтанного излучения как канала декогеренции. Обсуждаются некоторые аспекты системы, такие как динамика численности, эффект расстройки и появление электромагнитно-индуцированной прозрачности, контролируемой туннелированием. Полученные результаты показывают, что эффективностью туннельного взаимодействия и, как следствие, заселением непрямого экситонного состояния можно управлять путем настройки внешних физических параметров. Также показано, что при определенном выборе этих параметров непрямое экситонное состояние устойчиво к спонтанному излучению. Наконец, определяются диапазон частот приложенного импульса, в котором возникает туннельная прозрачность, и зависимость этого диапазона от значения параметра туннельной связи.

Манипулирование и динамическое управление квантовыми состояниями под действием когерентного излучения в последнее время стало одной из важнейших задач физики конденсированного состояния. В частности, полупроводниковые туннельно-связанные квантовые точки были отмечены как перспективная система динамического управления квантовыми состояниями. Это связано с тем, что можно сконструировать такие четко определяемые переходы в возбужденное состояние, как в искусственных атомах, и манипулировать их энергетическими уровнями, чтобы настроить количество ограниченных состояний и их энергетическое расстояние между подуровнями. В этой работе [4] авторы интересуются изучением динамики, связанной с двумя асиммет-

ричными туннельно-связанными квантовыми точками: такая модель хорошо известна как модель «квантовой молекулы». Эту систему можно описать следующим образом: изначально внутри обеих квантовых точек нет носителя заряда. Обозначим эту ситуацию как состояние $|0\rangle$. При приложении оптического импульса с электрическим полем E и частотой ω создается прямой экситон (электрон и дырка, связанные кулоновским взаимодействием) внутри первой квантовой точки. Эта ситуация соответствует состоянию $|1\rangle$. Прикладывая дополнительное напряжение, можно манипулировать профилями проводимости и валентной зоны. Это означает, что существует вероятность туннелирования электрона из первой квантовой точки во вторую и образуется непрямой экситон, который обозначили как состояние $|2\rangle$ [4].

В работе «Туннелирование в контактах с двойными барьерами и “горячими точками”» [5] авторы исследовали электронный транспорт в эпитаксиальных двойных магнитных туннельных переходах Fe(100)/MgO/Fe/MgO/Fe с мягким пробоем барьера («горячими точками»). Спецификой этих переходов являются сплошной средний слой и легирование азотом барьеров MgO, что обеспечивает мягкий пробой при смещениях около 0,5 В. В переходах с «горячими точками» наблюдаются квазипериодические изменения сопротивления в зависимости от напряжения смещения, что указывает на образование состояний квантовых ям, находящихся в среднем сплошном свободном слое Fe. Колебания комнатной температуры наблюдались как в параллельной, так и в антипараллельной магнитной конфигурации и для обеих поляризаций смещения. Для качественного объяснения этого эффекта предложена простая модель туннелирования через «горячие точки» в двухбарьерном магнитном переходе.

В работе «Джозефсоновский транзистор сверхмалой диссипации» [6] представлен транзистор на основе сверхпроводника – нормального металла –сверхпроводника (SNS) с использованием сверхпроводящих микроохладителей. Предлагаемое 4-концевое устройство состоит из длинного SNS-джозефсоновского перехода, N-область которого дополнительно симметрично соединена со сверхпроводящими резервуарами через туннельные барьеры (I). Приложенное напряжение линии SINIS позволяет изменять температуру квазичастиц в слабой связи, управляя тем самым туннельным джозефсоновским током с диссипацией. Авторы [6] показывают, что в подходящих режимах напряжения и температуры можно достичь значительного усиления туннельного диссипативного сверхтока по отношению к равновесию из-за электронного «охлаждения», генерируемого управляющим напряжением. Чрезвычайно низкое рассеивание мощности, присущее конструкции, делает это устройство подходящим для ряда электронных приложений.

Особый интерес представляют наблюдаемые эффекты диссипативного туннелирования при исследовании управляемого осциллирующего режима фотопроводимости в синтезированных гетероструктурах с вертикально-туннельно-связанными самоформирующимися квантовыми точками InAsGaAs (00l).

Результаты экспериментальных исследований фотоэлектрических свойств GaAs*p-i-n* фотодиода с двойными асимметричными квантовыми точками (ДАКТ) InAs были получены методом самоформирования в процессе МОС-гидридной эпитаксии (ННГУ им. Н. И. Лобачевского) [7]. В зависимости фототока от напряжения обратного смещения при монохроматическом фотовозбуждении ДАКТ на длине волны, соответствующей энергии межзонных оптических переходов между основными состояниями дырок и электронов в КТ большего размера, обнаружены три пика, связанных с туннелированием фотовозбужденных электронов между КТ, в том числе – диссипативного (с поглощением и испусканием оптических фононов). *Результаты эксперимента качественно согласуются с теоретической полевой зависимостью вероятности 1D-диссипативного туннелирования между КТ* [7].

В ряде приложений оказывается важным использовать синтезированные гетероструктуры с двойными вертикально-туннельно-связанными самоформирующимися квантовыми точками

InAsGaAs (00l), когда реализуется режим 2D-диссипативного туннелирования с эффектами 2D-бифуркаций и квантовых биений. При этом рассматривается модель 2D-диссипативного туннелирования в условиях внешнего электрического поля при конечной температуре с учетом влияния двух локальных фононных мод матрицы среды – термостата и вычисляется вероятность 2D-диссипативного туннелирования, которая с экспоненциальной точностью определяет величину туннельного тока.

При исследовании полевых и температурных зависимостей вероятности 2D-диссипативного туннелирования в пределе сильной диссипации с учетом влияния двух локальных фононных мод матрицы среды – термостата получены следующие теоретические результаты.

На рис. 1 представлена полевая зависимость вероятности 2D-параллельного синхронного диссипативного туннелирования, рассчитанная с учетом влияния двух локальных фононных мод. В режиме синхронного параллельного переноса туннелирующих частиц наличие двух локальных фононных мод приводит к появлению двух устойчивых пиков на указанной полевой зависимости. Из рис. 1 видно, что расстояние между пиками зависит от температуры и возрастает с ростом температуры. Минимум между двумя пиками соответствует случаю симметричного двухъямного модельного потенциала и отвечает режиму блокировки туннелирования при существенном влиянии двух локальных фононных мод. Если взаимодействие с локальными фононными модами «выключить», то вместо блокировки туннелирования для симметричного двухъямного потенциала будет иметь место единичный пик на кривой вероятности туннелирования при одной из полярностей внешнего электрического поля. Минимум отвечает малому, но ненулевому значению вероятности туннелирования (рис. 2).

Рисунок 3 показывает, что изменение параметра взаимодействия α^* туннелирующих частиц слабо влияет на вероятность 2D-параллельного синхронного диссипативного туннелирования.

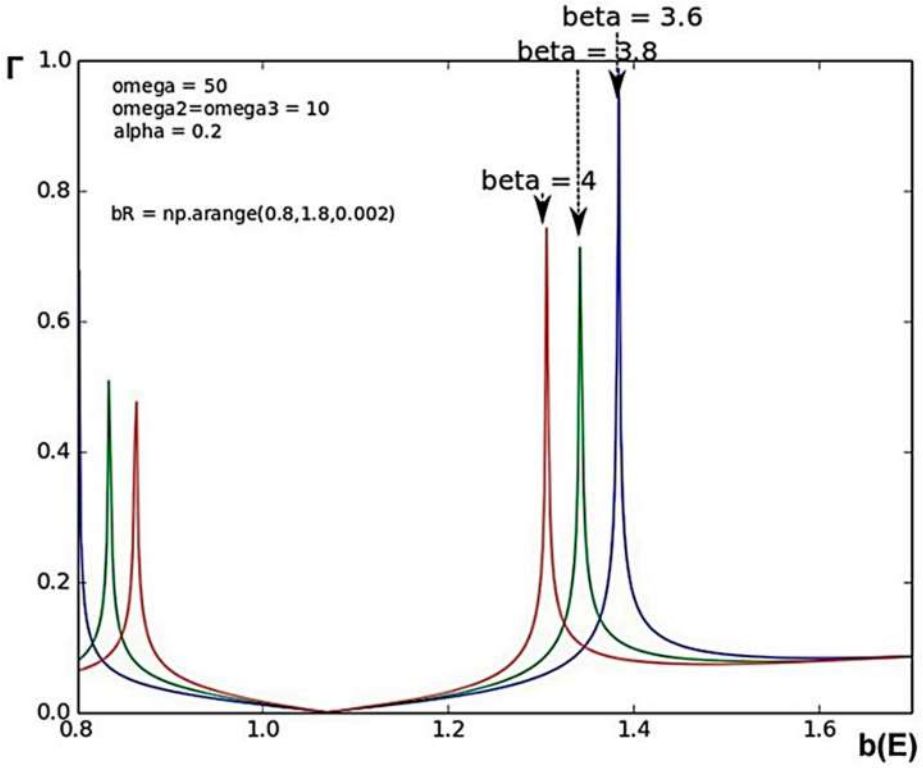


Рис. 1. Полевая зависимость вероятности 2D-параллельного синхронного диссипативного туннелирования с учетом влияния двух локальных фононных мод в случае, когда частота модельного осцилляторного потенциала в пять раз превосходит частоту локальных фононных мод

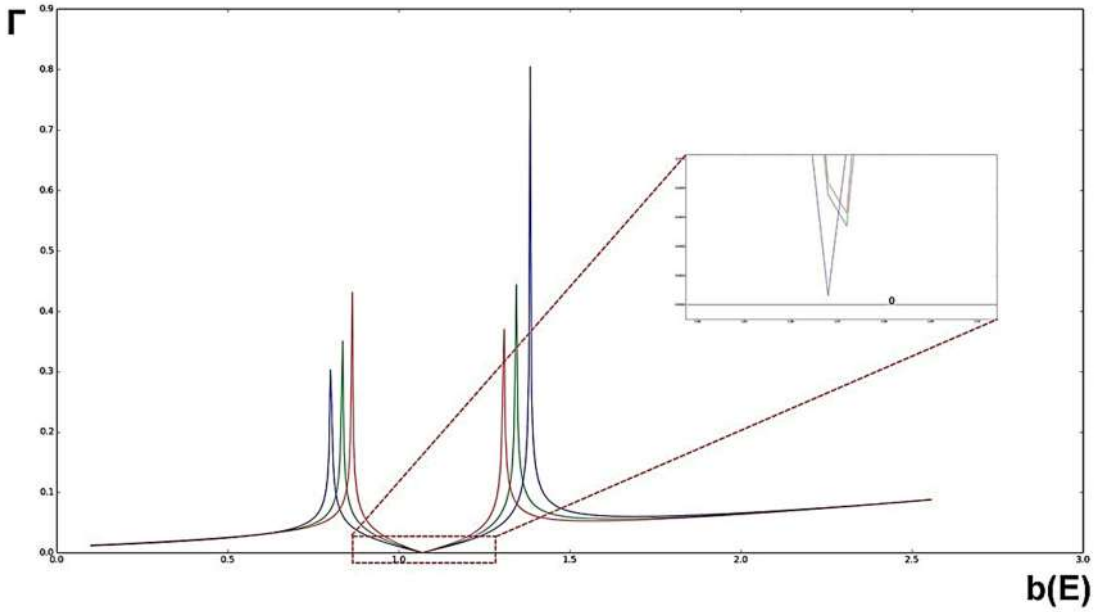


Рис. 2. Режим блокировки 2D-параллельного синхронного диссипативного туннелирования

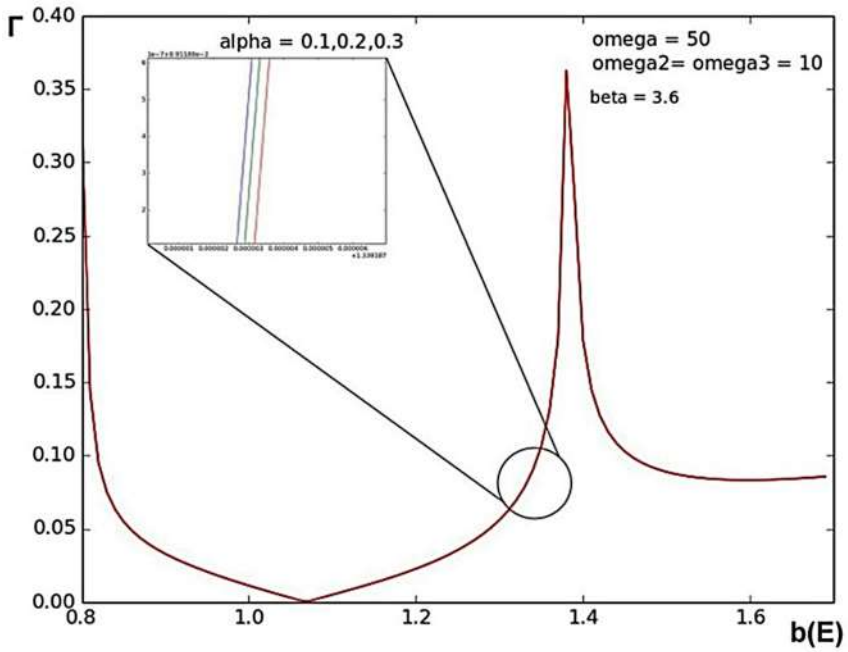


Рис. 3. Влияние параметра взаимодействия параллельно туннелирующих частиц
в синхронном режиме переноса на вероятность
2D-параллельного синхронного диссипативного туннелирования

Соотношение между высотами левого и правого пиков на полевой зависимости вероятности 2D-параллельного синхронного диссипативного туннелирования зависит от соотношения частот локальных фононных мод и частоты двухъямного осцилляторного потенциала вдоль параллельных координат туннелирования (см. рис. 1, 4). На рис. 1 высоты правых пиков выше, чем левых, а на рис. 4 имеет место обратная ситуация. При этом для случая, представленного на рис. 1, частота модельного потенциала выбиралась в пять раз больше частот локальных фононных мод, а для рис. 4 эти частоты были сравнимыми.

Из рис. 5 видно, что на теоретической кривой температурной зависимости вероятности 2D-диссипативного туннелирования с уменьшением напряженности внешнего электрического поля, начиная с некоторых пороговых значений, один из двух устойчивых пиков, соответствующих взаимодействию туннелирующих частиц с двумя локальными фононными модами, может расщепляться на два, что, по-видимому, связано с эффектом «подстройки» стартового энергетического уровня под состояния, обусловленные электрон-фононным взаимодействием.

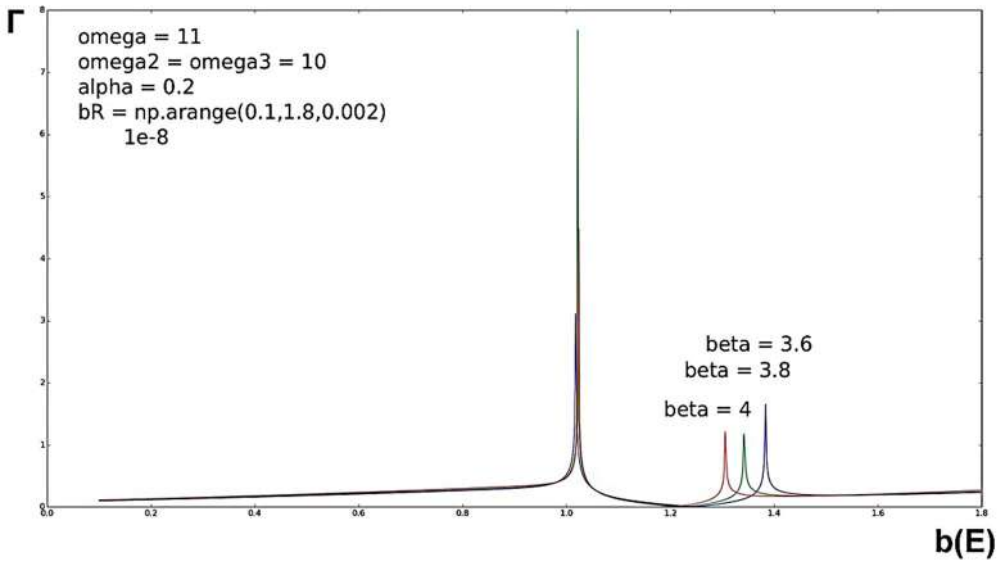


Рис. 4. Полевая зависимость вероятности 2D-параллельного синхронного диссипативного туннелирования в случае, когда частота модельного осцилляторного потенциала в 1,1 раза превосходила частоты локальных фононных мод

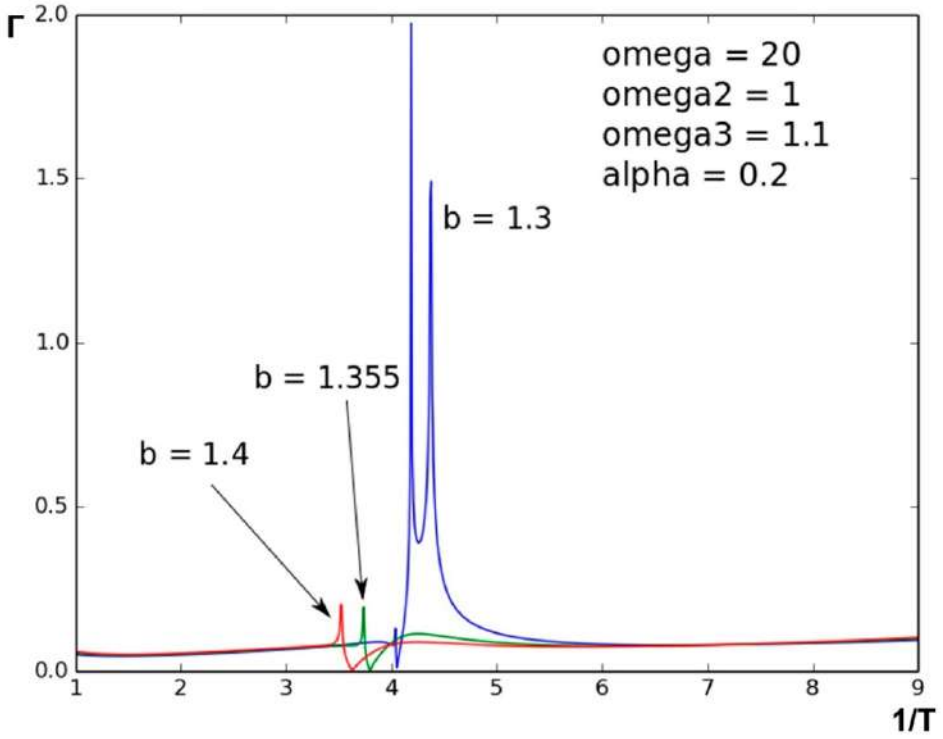


Рис. 5. Температурная зависимость вероятности 2D-диссипативного туннелирования

Полученная полевая зависимость вероятности 2D-диссипативного туннелирования с учетом влияния двух локальных фононных мод позволяет проанализировать режим 2D-туннельных бифуркаций (смена режима туннелирования с синхронного на асинхронный), а также квантовых биений в окрестности точки бифуркации. Так, на рис. 6 после режима синхронного параллельного туннельного переноса с двумя характерными пиками точка излома отвечает точке бифуркации, а последующие осцилляции – квантовым биениям.

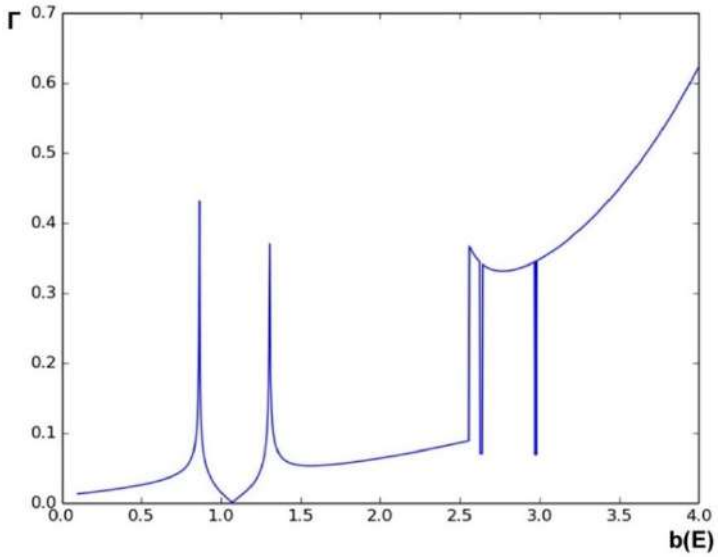


Рис. 6. Полевая зависимость вероятности 2D-диссипативного туннелирования с учетом точки бифуркации и режима квантовых биений

Наряду с режимом квантовых биений с «провалами» на полевой зависимости вероятности 2D-диссипативного туннелирования (см. рис. 6) при увеличении температуры и при частотах фононных локальных мод, значительно меньших характерных частот двухъямного осциллятора, может иметь место режим квантовых биений с «резонансной» структурой (рис. 7), связанной с интерференцией различных каналов туннелирования [1].

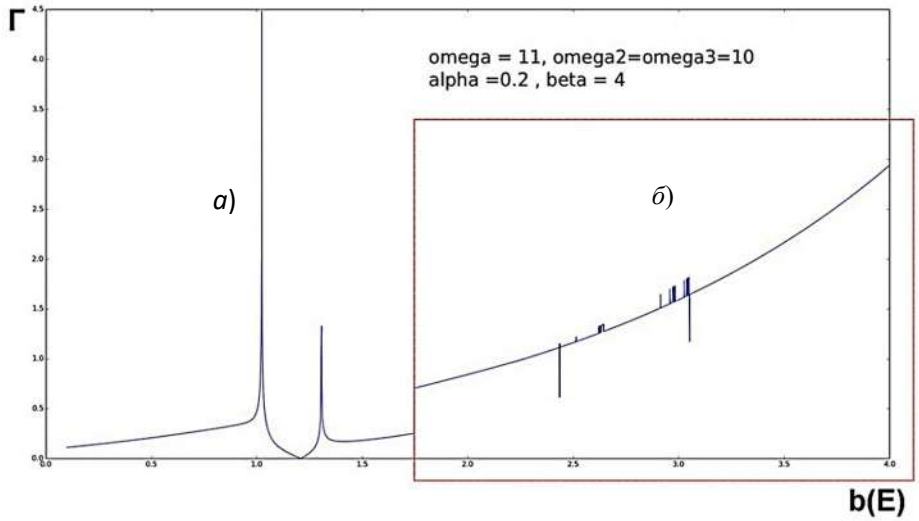


Рис. 7. Режим квантовых биений на полевой зависимости вероятности 2D-диссипативного туннелирования:
а – «резонансы» до точки бифуркации; *б* – «резонансы» и «провалы» выше точки бифуркации

Подобные режимы квантовых биений напоминают особенности туннельной проводимости для полупроводниковых наноструктур с примесными атомами – типа резонансов Фано, возникающих из-за интерференции между резонансным и нерезонансным каналами туннелирования [8].

Кроме того, теоретически предсказан возможный экспериментально наблюдаемый эффект 2D-диссипативного туннелирования с учетом влияния двух локальных фононных мод матрицы среды – термостата на полевой зависимости интенсивности фотолюминесценции на примере синтезированных гетероструктур с двойными вертикально-туннельно-связанными самоформирующимися квантовыми точками InAsGaAs (00l), когда реализуется режим 2D-диссипативного туннелирования с эффектами 2D-бифуркаций и квантовых биений (рис. 8).

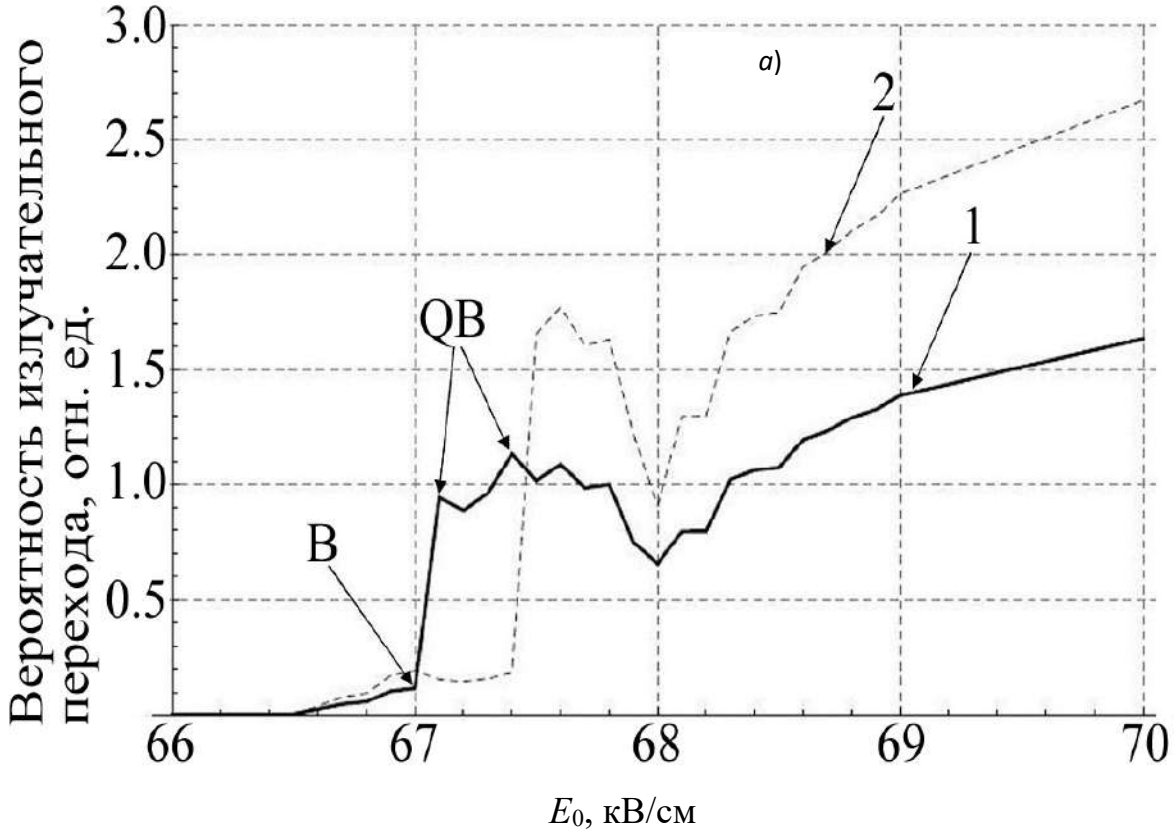
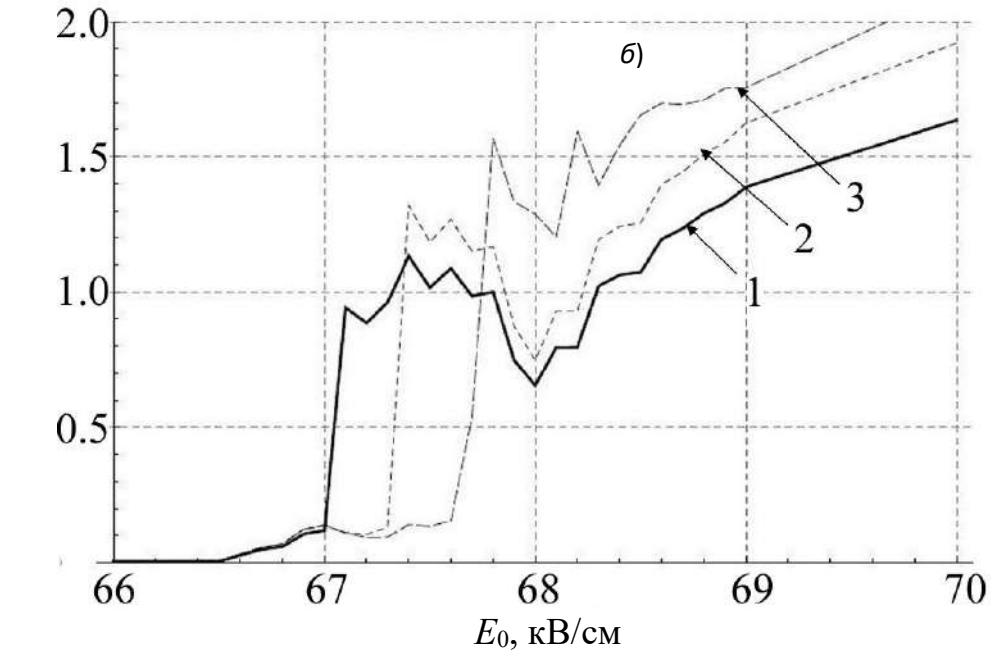


Рис. 8. Зависимость вероятности излучательного перехода ВИП электрона между квазистационарными *u*- и *g*-состояниями в InSb-КТ с D_2^--центром от величины напряженности внешнего электрического поля E_0 при величинах радиуса КТ $R_0 = 50$ нм, высоты потенциального барьера $U_0 = 0,35$ эВ, энергии фотона $\hbar\omega = 5,3$ мэВ, расстоянии между примесными центрами $\rho_{12} = 4,8$ нм, значениями параметров диссипативного туннелирования в боровских единицах: температурного $\varepsilon_T^* = 1,3$, $\varepsilon_C^* = 2,2$, для различных значений «вязкого» параметра ε_C^* 2D-диссипативного туннелирования (с участием двух локальных фононных мод, частоты которых определены параметрами ε_{L1}^* и ε_{L2}^*): *а* – 1 – $\varepsilon_T^* = 1,3$, $\hbar\omega = 5,22$ мэВ; 2 – $\varepsilon_T^* = 1,5$, $\hbar\omega = 5,41$ мэВ; *б* – 1 – $\varepsilon_{L1}^* = 1$, $\varepsilon_{L2}^* = 1$; $\hbar\omega = 5,22$ мэВ; 2 – $\varepsilon_{L1}^* = 1,4$, $\varepsilon_{L2}^* = 1,6$; $\hbar\omega = 5,31$ мэВ; 3 – $\varepsilon_{L1}^* = 1,6$, $\varepsilon_{L2}^* = 1,8$; $\hbar\omega = 5,59$; мэВ; *в* – 1 – $\varepsilon_C^* = 2,2$; $\hbar\omega = 5,22$ мэВ; 2 – $\varepsilon_C^* = 2,5$; $\hbar\omega = 5,16$ мэВ. («В» – точка бифуркации; «QB» – область квантовых биений) (начало)



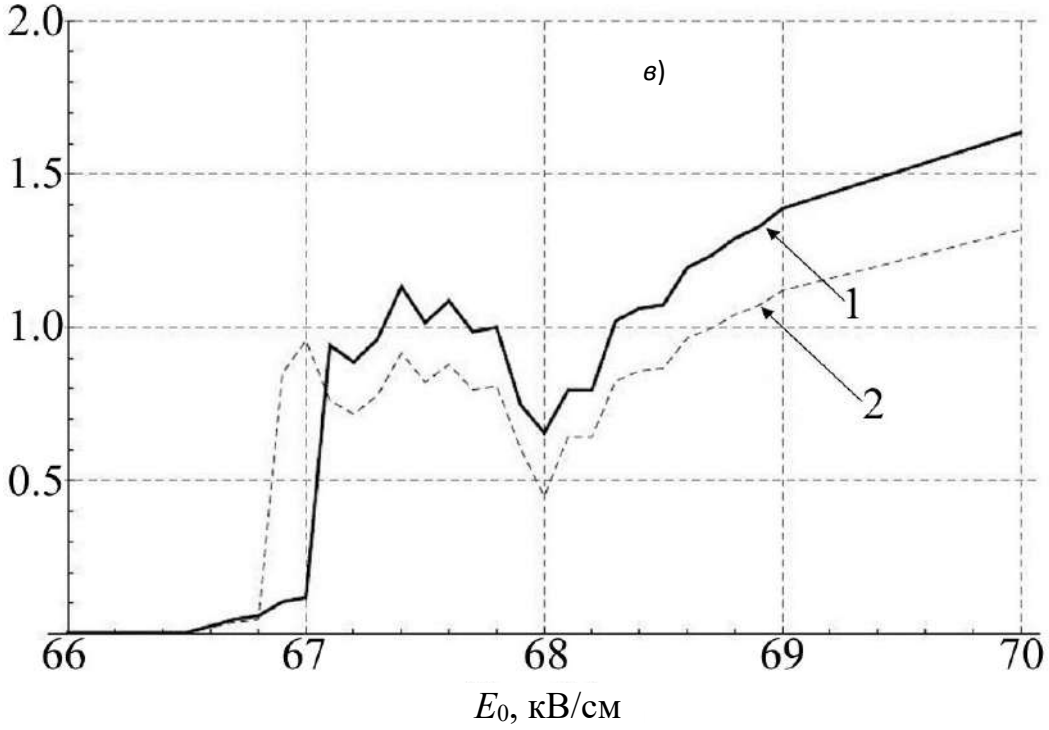


Рис. 8. Окончание

Таким образом, в одноинстантонном приближении аналитически найдена вероятность 2D-параллельного диссипативного туннелирования с точностью до экспоненты, определяемой квазиклассическим действием, в модели 2D-осцилляторного потенциала при конечной температуре во внешнем электрическом поле с учетом влияния двух локальных фононных мод.

Показано, что в режиме синхронного параллельного переноса туннелирующих частиц наличие двух локальных фононных мод приводит к появлению двух устойчивых пиков на полевой зависимости вероятности 2D-диссипативного туннелирования.

Установлено, что с уменьшением напряженности внешнего электрического поля, начиная с некоторых пороговых значений, один из устойчивых пиков на температурной зависимости вероятности 2D-диссипативного туннелирования расщепляется на два, что связано с эффектом «подстройки» стартового энергетического уровня под состояния, обусловленные электрон-фононным взаимодействием.

Теоретически предсказан эффект 2D-бифуркаций и квантовых биений двух типов (осцилляции и «квантовая гребенка») на туннельных вольт-амперных характеристиках (ВАХ), а также на полевой зависимости интенсивности фотолюминесценции в режиме сильной диссипации с учетом влияния двух локальных фононных мод на примере синтезированных гетероструктур с двойными вертикально-туннельно-связанными самоформирующимися квантовыми точками InAsGaAs (00l), когда реализуется режим 2D-диссипативного туннелирования с эффектами 2D-бифуркаций и квантовых биений.

Список литературы

1. Supriya Karmakar, Faquir C. Jain. Future Semiconductor Devices for Multi-Valued Logic Circuit Design // Materials Sciences and Applications. 2012. № 3. P. 807–814. URL: http://www.SciRP.org/journal/msa

2. Eliade Stefanescu, Werner Scheid. Superradiant dissipative tunneling in a double p–i–n semiconductor heterostructure with thermal injection of electrons // Physica. 2007. Vol. A 374. P. 203–210.

3. Nan Ma and Debdeep Jena. Interband tunneling in two-dimensional crystal semiconductors // APPLIED PHYSICS LETTERS. 2013. Vol. 132, № 102. P. 1–5.

4. Borges H. S., Sanz L., Alcalde A. M. Dissipative dynamics in coupled quantum dots: control of tunneling and electromagnetically induced transparency // AIP Conf. Proc. 2010. P. 377–378. URL: https://doi.org/10.1063/1.3295459

5. Herranz D., Aliev F. G., Tiusan C. [et al.]. Tunneling in Double Barrier Junctions with “Hot Spots”// Physical Review Letters. 2010. Vol. 105. P. 47–207. doi: 10.1103/PhysRevLett.105.047207

6. Семёнов М. Б., Кревчик В. Д., Филатов Д. О., Шорохов А. В. [и др.]. Диссипативное туннелирование электронов в вертикально связанных двойных асимметричных квантовых точках InAs/GaAs (001) // Журнал технической физики. 2021. Т. 91. Вып. 10. С. 1431–1440. doi: 10.21883/JTF.2021.10.51354.66-21

7. Манцевич В. Н. Неравновесные эффекты и нестационарный электронный транспорт в полупроводниковых наноструктурах с межчастичным взаимодействием : дис. … д-ра физ.-мат. наук. М. : МГУ им. М. В. Ломоносова, 2014. 337 с.

8. Mantsevich V. N., Maslova N. S. Spatial effects of Fanoc resonance in local tunnelingc conductivity in vicinity of impurity on semiconductorc surface // JETP Lett. 2010. Vol. 91, № 3. P. 139–142.

9. Maltezopoulos T., Bolz A., Meyer C., Heyn C. [et al.]. Wave-Function Mapping of InAs Quantum Dots by Scanning Tunneling Spectroscopy // Phys. Rev. Lett. 2003. Vol. 91, № 19. P. 196804-1–196804-4.

Информация об авторах

Кревчик Владимир Дмитриевич, доктор физико-математических наук, профессор, заведующий кафедрой «Физика», декан факультета «Информационные технологии и электроника», Пензенский государственный университет.

Семёнов Михаил Борисович, доктор физико-математических наук, профессор, профессор-консультант кафедры «Физика», Пензенский государственный университет.

Кревчик Павел Владимирович, магистрант, Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.

УДК 538.9

ЭФФЕКТЫ ДИССИПАТИВНОГО ТУННЕЛИРОВАНИЯ В СТРУКТУРАХ С ЗОЛОТЫМИ НАНОЧАСТИЦАМИ

В. Д. Кревчик[1], М. Б. Семёнов[2], П. В. Кревчик[3]

[1, 2, 3]Пензенский государственный университет, Пенза, Россия

[1, 2, 3]physics@pnzgu.ru

Аннотация. Рассмотрена модель 2D-диссипативного туннелирования в пределе слабой диссипации для планарной структуры с золотыми наночастицами под иглой кантилевера совмещенного атомного силового и сканирующего туннельного микроскопа, когда коэффициент взаимодействия туннелирующих электронов по параллельным координатам реакции может сменить знак. Показано, что в этом случае наблюдается теоретически и подтверждается экспериментально эффект двойных 2D-бифуркаций и квантовых биений.

Ключевые слова: диссипативное туннелирование, металлические наночастицы, эффект бифуркаций, квантовые биения

Для цитирования: Кревчик В. Д., Семёнов М. Б., Кревчик П. В. Эффекты диссипативного туннелирования в структурах с золотыми наночастицами // Вестник Пензенского государственного университета. 2024. № 4. С. 160–166.

В последние годы заметно возросло количество приборных приложений в области современной наноэлектроники с управляемыми характеристиками на основе туннельного эффекта. В частности, речь может идти о создании туннельных диодов [1], сенсоров [2], преобразователей терагерцового в ИК-излучение [3] и других устройств туннельной наноэлектроники на основе золотых наночастиц. Так, например, в работе «Преобразователи терагерцового излучения в инфракрасное для визуализации рака кожи человека: проблемы и возможности» коллектива авторов [3] рассматриваются преобразователи терагерцового в инфракрасное (ТГц – ИК) излучение, которые могут визуализировать рак кожи человека, преобразуя его специфические контрастные паттерны, распознаваемые в диапазоне ТГц-излучения, в ИК-паттерны, обнаруживаемые стандартной ИК-камерой. В основе предлагаемых преобразователей ТГц – ИК лежат плоские матрицы, прозрачные как в визуализируемом ТГц-диапазоне, так и в рабочем диапазоне ИК-камеры; эти матрицы содержат внедренные металлические (золотые) наночастицы (рис. 1), которые при облучении ТГц-лучами преобразуют энергию ТГц-фотонов в тепло и становятся наноисточниками ИК-излучения, обнаруживаемыми ИК-камерой.

При этом для обеспечения синтеза наиболее эффективного размера внедренных золотых наночастиц может оказаться востребованным метод их контролируемого роста в системе совмещенного атомного силового и сканирующего туннельного микроскопа с использованием эффектов диссипативного туннелирования [4].

Золотые наночастицы в настоящее время используются в различных устройствах наноэлектроники и нанофотоники с управляемыми характеристиками.

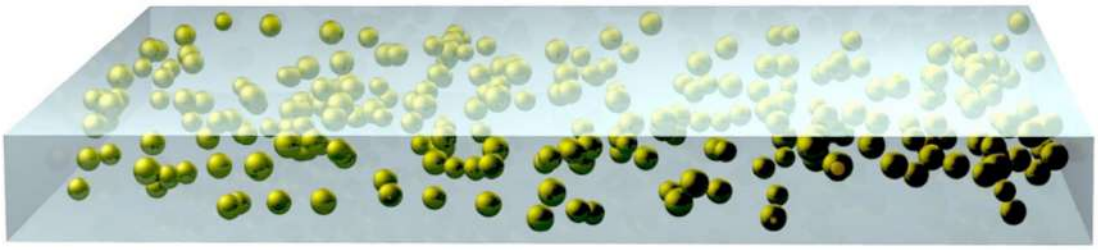


Рис. 1. «Рабочий элемент» преобразователя ТГц – ИК:
матрица с внедренными металлическими (золотыми) наночастицами [3]

Так, например, в работе «Усиление взаимодействия света и материи в двумерных материалах с помощью оптических микро/наноархитектур для высокопроизводительных оптоэлектронных устройств» [5] показано, что двумерные материалы с использованием Au-наночастиц являются многообещающим решением для электронных и оптоэлектронных устройств следующего поколения благодаря своим уникальным свойствам (рис. 2).

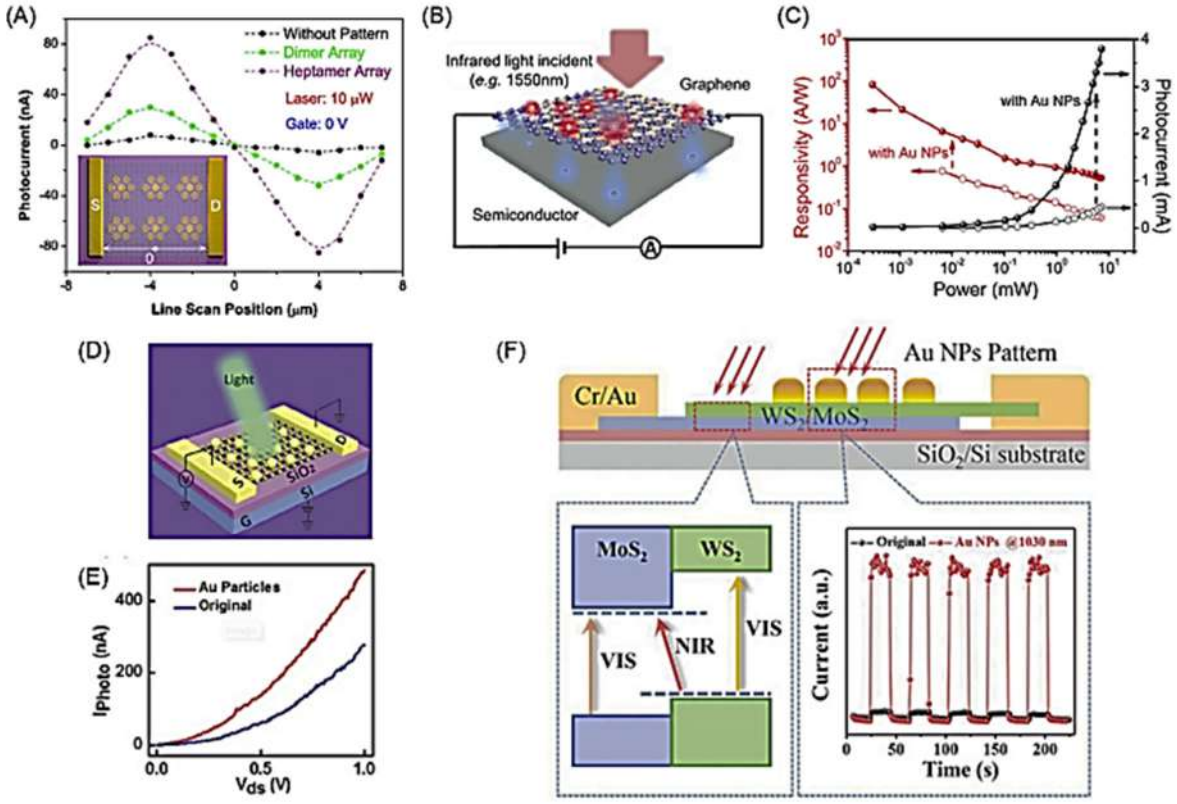


Рис. 2. Двумерные материал-плазмонные гибридные фотодетекторы [5]:
A – измерения фототока фотодетектора «сэндвича» графен – антенна, полученного вдоль направления строчной развертки с различной геометрией Au-антенны. Данные с устройства управления без Au-антенны также представлены. На вставке – схема устройства с указанием нулевой точки. B – схема инфракрасного фотодетектора из графена/кремния, связанного с Au–NP. C – фотоответ устройства графен/кремний при различной мощности освещения (1550 нм) с НЧ–Au и без них. D – схема фотодетектора MoS_2 с несколькими слоями, соединенного с периодической Au-наноматрицей. E – зависимость фототока от напряжения на стоке устройства с несколькими слоями MoS_2 с Au-наноматрицами и без них. F – схема фотодиода WS_2 /MoS_2, связанного с Au–NP, и процессы поглощения для видимого и ближнего инфракрасного света, а также для переключения фотооткликов устройства с плазмонной связью и без нее на длине волны 1030 нм освещения [5]

Из-за атомной толщины 2D-материалов длина взаимодействия света с веществом в 2D-материалах намного меньше, чем в объемных материалах, что ограничивает возможности оптоэлектронных устройств, состоящих из 2D-материалов. Для улучшения взаимодействия света и материи были введены оптические микро/наноархитектуры в двумерные оптоэлектронные устройства.

Авторы [5] обсуждают различные стратегии улучшения взаимодействия света и вещества в двумерных материалах, а именно: плазмонного эффекта, волновода, оптического резонатора и архитектуры отражения. Также описаны текущие достижения в области высокопроизводительных оптоэлектронных устройств из 2D-материалов (например, фотодетекторов, электрооптических модуляторов, светоизлучающих диодов и молекулярных сенсоров), которым помогают эти стратегии усовершенствования. В заключительной части статьи авторы обсудили будущие проблемы и возможности проектирования микро/нанофотонных структур в устройствах из 2D-материалов.

Эффекты 2D-туннельных бифуркаций в рамках науки о квантовом туннелировании с диссипацией впервые были предсказаны в работе Б. И. Ивлева и Ю. Н. Овчинникова (ИТФ им. Л. Д. Ландау) для планарных структур взаимодействующих контактов Джозефсона в виде характерных изломов на соответствующих туннельных ВАХ [6]. Предсказанные теоретически изломы до настоящего момента не удалось выявить экспериментально из-за наличия существенных шумов в окрестности точки бифуркации. В рамках рассматриваемой нами теоретической модели 2D-диссипативного туннелирования в 2D-осцилляторном потенциале в одноинстантонном квазиклассическом приближении в пределе слабой диссипации при конечной температуре в условиях внешнего электрического поля в системе совмещенного атомного силового и сканирующего туннельного микроскопа для планарной структуры с отдельными золотыми наночастицами в случае обычной диэлектрической матрицы удалось теоретически выявить единичный излом (точку бифуркации) на полевой зависимости вероятности 2D-диссипативного туннелирования. Аналогичный излом (точку бифуркации) экспериментально выявили для исследуемых структур на отдельных туннельных ВАХ в лаборатории зондовой микроскопии ННГУ им. Н. И. Лобачевского [6]. Оказалось, что на отдельных туннельных ВАХ, когда туннельный ток протекал по параллельным каналам через соседние золотые наночастицы, а не через одну отдельную наночастицу, удалось экспериментально наблюдать не один, а два характерных излома, т.е. двойные 2D-туннельные бифуркации. Кроме того, упорядоченные планарные структуры с золотыми туннельно-связанными наночастицами в диэлектрической матрице при определенных условиях могут обладать свойствами метаматериала, т.е. эффективной отрицательной относительной диэлектрической проницаемостью. Следовательно, актуальной целью стало теоретическое исследование двойных 2D-туннельных бифуркаций и квантовых биений в системе «игла кантилевера совмещенного АСМ/СТМ – КТ(КМ)», моделируемой двухъямным осцилляторным потенциалом при наличии внешнего электрического поля при конечной температуре, в случае матрицы, обладающей свойствами метаматериала.

Результаты проведенных экспериментов свидетельствуют о возможности экспериментального наблюдения устойчивых 2D-туннельных бифуркаций с диссипацией.

В случае позиционирования острия АСМ зонда между двумя близко расположенными НЧ Au (рис. 3) появляется возможность параллельного туннелирования электронов между АСМ зондом и подложкой одновременно через две НЧ Au (квазидвумерный случай). При этом на туннельных ВАХ наблюдалась пара изломов, соответствующих двойному эффекту бифуркаций 2D-туннелирования (рис. 4). Помимо двойных изломов, на ВАХ наблюдались квантовые биения в окрестности точек двумерных бифуркаций.

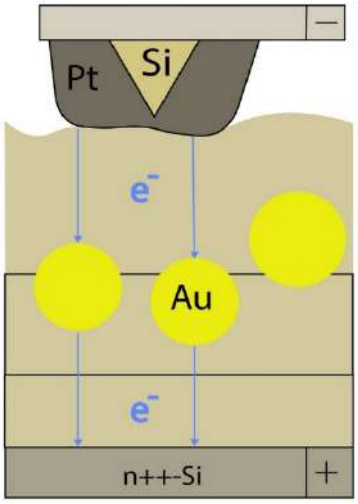
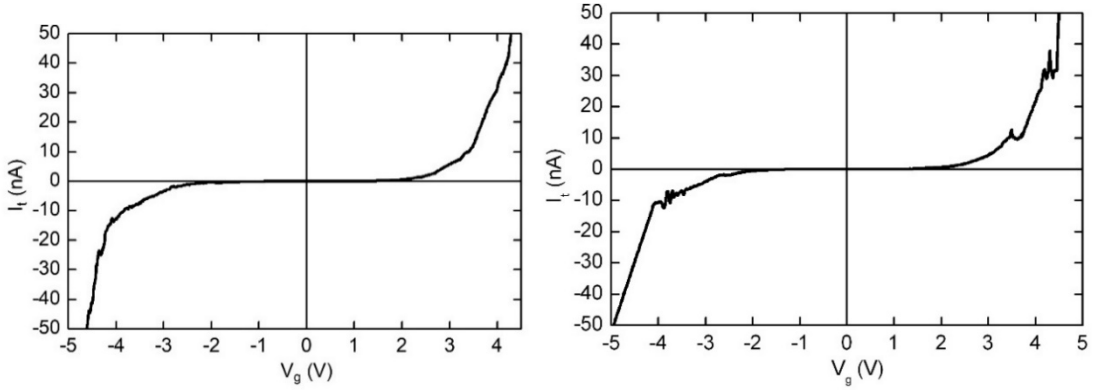


Рис. 3. Схема параллельного туннелирования электронов между АСМ зондом
и проводящей подложкой через две НЧ Au
при исследовании диссипативного туннелирования электронов
через НЧ Au в туннельно-прозрачной пленке SiO_2/n^+-Si методом туннельной АСМ



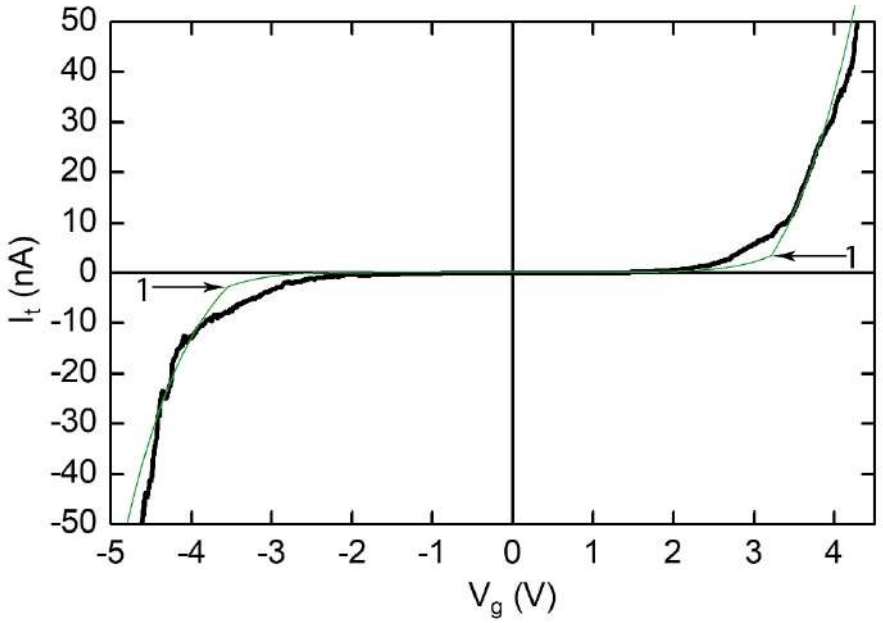
а) *б*)

Рис. 4. Примеры ВАХ контакта АСМ зонда к структуре
SiO_2(1,5 нм)/SiO_2:nc-Au(1,6 нм)/SiO_2(1,8 нм)/n^+-Si(001),
измененных при позиционировании АСМ зонда между двумя НЧ Au:
а – со «сглаженными» двойными изломами (без квантовыми биений); *б* – с квантовыми биениями

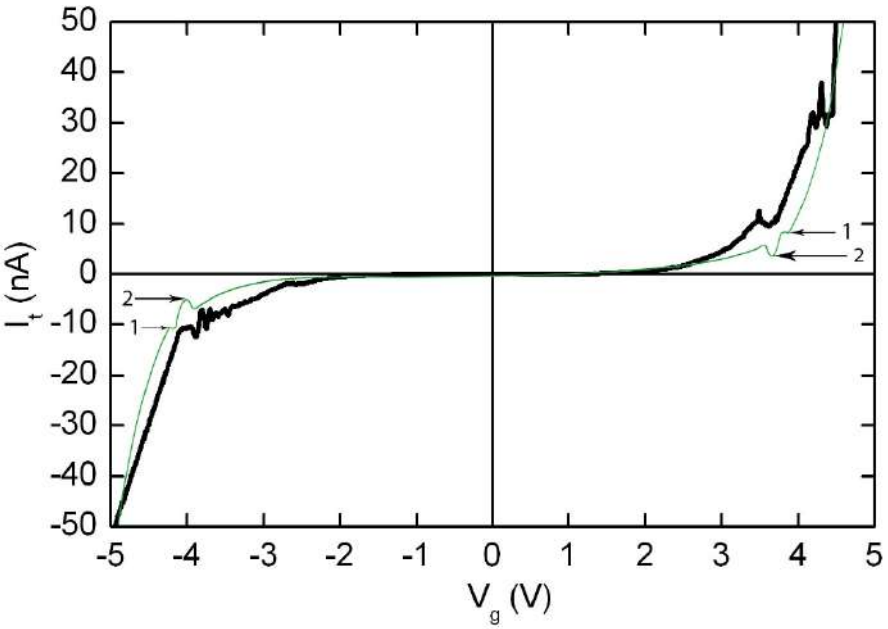
Теоретический расчет 2D-бифуркационных мод проводился в модельном потенциале 2D-осциллятора при конечной температуре во внешнем электрическом поле в пределе слабой диссипации, т.е. считалось, что взаимодействие электронов, туннелирующих между НЧ, с матрицей-термостатом, обеспечивающее реальный переход в состояния, локализованные в соседней НЧ, достаточно мало, но достаточно для обеспечения затухания потенциала осциллятора двойной ямы, используемого в предлагаемой модели [7].

В случае позиционирования острия АСМ зонда между двумя близкорасположенными Au (квазидвумерная система), как отмечено выше, на полевой зависимости вероятности 2D-диссипативного туннелирования имеется не один, а два характерных излома. Результаты сравнения расчетных полевых зависимостей вероятности 2D-туннелирования с экспериментальным ВАХ квазидвумерных систем (рис. 5) также показывают удовлетворительное качественное соответствие результатов расчетов и экспериментов. При этом кривые на рис. 5,*а* соответствуют двумер-

ным бифуркациям без квантовых биений. На рис. 5,*б* в окрестности точек двумерных бифуркаций также наблюдались квантовые биения.



а)



б)

Рис. 5. Сравнение экспериментальных туннельных ВАХ
для случая позиционирования АСМ зонда между двумя близкорасположенными НЧ Au (чёрная кривая)
с расчетными полевыми зависимостями вероятности 2D-диссипативного туннелирования
(зелёная кривая), рассчитанными в пределе слабой диссипации

С использованием метода самоформирования НЧ Au в процессе окисления в кислородной плазме тлеющего разряда тонких пленок аморфной смеси Au–Si (1 : 7), осажденных на подложки n^+-Si(001) методом импульсного лазерного осаждения (ИЛО), сформированы 2D-массивы НЧ Au

с латеральными размерами 2–5 нм и толщиной 1,0–1,5 нм, встроенные в туннельно-прозрачные (толщиной порядка 5 нм) пленки SiO_2/n^+-Si(001).

Методом туннельной АСМ изучен поперечный туннельный транспорт электронов в пленках SiO_2:nc-Au/n^+-Si(001). На токовых изображениях пленок обнаружены участки локального увеличения силы тока через АСМ зонд размером порядка 10 нм, связанные с туннелированием электронов между АСМ зондом с Pt покрытием и n^+-Si подложкой через индивидуальные НЧ Au.

С целью изучения процессов диссипативного туннелирования электронов через НЧ Au, внедренные в пленку SiO_2, измерены ВАХ контакта АСМ зонда к пленке SiO_2:nc-Au/n^+-Si(001) при различных положениях острия АСМ зонда относительно НЧ Au. В случае позиционирования острия АСМ зонда строго над индивидуальной НЧ Au (квазиодномерный случай) на туннельной ВАХ наблюдается единичный излом, связанный с бифуркацией 2D-диссипативного туннелирования электронов между АСМ зондом с двумя выступами на острие и n^+-Si подложкой через НЧ Au. В случае позиционирования АСМ зонда между двумя близко расположенными НЧ Au (квазидвумерный случай) на ВАХ обнаружены двойные изломы, связанные с двойным эффектом бифуркаций при диссипативном 2D-туннелировании электронов между АСМ зондом и n^+-Si подложкой параллельно через две близко расположенные НЧ Au.

В туннельных ВАХ контакта АСМ зонда к пленкам SiO_2:nc-Au/n^+-Si(001) обнаружены особенности вблизи точек 2D-туннельных бифуркаций, связанные с предсказанными ранее теоретически квантовыми осцилляциями при диссипативном 2D-туннелировании.

Проведен теоретический расчет вероятности 2D-диссипативного туннелирования в модельном потенциале 2D-осциллятора при конечной температуре во внешнем электрическом поле в пределе слабой диссипации с учетом 2D-мод бифуркаций для квазиодномерного и квазидвумерного случаев. Проведено сравнение результатов расчетов и экспериментальных туннельных ВАХ контакта АСМ зонда к поверхности пленки SiO_2:nc-Au/n^+-Si(001). В первом случае получено качественное согласие экспериментальных ВАХ и расчетных полевых зависимостей в модельном потенциале 2D-осциллятора для случая параллельного 2D-диссипативного туннелирования. Во втором случае качественное согласие между экспериментальными ВАХ и теоретической полевой зависимостью для вероятности 2D-диссипативного туннелирования получено в предположении отрицательной эффективной относительной диэлектрической проницаемости матрицы-термостата.

В результате:

– получена аналитическая формула вероятности 2D-диссипативного туннелирования в модельном 2D-осцилляторном потенциале в одноинстантонном квазиклассическом приближении в пределе слабой диссипации при конечной температуре в условиях внешнего электрического поля в системе совмещенного атомного силового и сканирующего туннельного микроскопа для планарной структуры с золотыми наночастицами в матрице, обладающей свойствами метаматериала;

– теоретически найдены эффекты двойных 2D-туннельных бифуркаций и квантовых биений на полевой зависимости вероятности 2D-диссипативного туннелирования в рассматриваемой модели для планарной структуры с золотыми наночастицами в матрице, обладающей свойствами метаматериала;

– получено качественное совпадение полученных теоретических зависимостей полевой зависимости вероятности 2D-диссипативного туннелирования с экспериментальными туннельными ВАХ для исследуемых структур, содержащими двойные изломы, интерпретируемые как двойные бифуркации, и квантовые биения в окрестности точек бифуркации.

Таким образом, исследования туннельных эффектов для систем с золотыми наночастицами являются актуальным направлением исследований современных устройств наноэлектроники и нанофотоники.

Список литературы

1. Hao-Tse Hsiao, I-Cheng Yao, I-Chih Ni, Shien-Der Tzeng [et al.]. Gold-Nanoparticle-Coated Ge MIS Photodiodes // Journal of the Electron Devices Society. 2018. Vol. 6. P. 95–99.

2. Guomei Zhang. Functional gold nanoparticles for sensing applications // Nanotechnol. Rev. 2013. № 2 (3). P. 269–288. doi: 10.1515/ntrev-2012-0088

3. Moldosanov K., Bykov A., Kairyev N., Khodzitsky M. [et al.]. Terahertz-to-infrared converters for imaging the human skin cancer: challenges and feasibility // Journal of Medical Imaging. 2023. № 10 (02). P. 23–501. doi: 10.1117/1.JMI.10.2.023501

4. Пат. на изобретение RU 2533533 C1 от 19.09.2014. Способ контролируемого роста квантовых точек из коллоидного золота / Кревчик В. Д., Семёнов М. Б., Артёмов И. И., Горшков О. Н.; опубл. 20.11.2014.

5. Li Tao, Zefeng Chen, Zhiyong Li, Jiaqi Wang. Enhancing light-matter interaction in 2D materials by optical micro/nano architectures for high-performance optoelectronic devices // InfoMat. 2020. № 3 (4). URL: https://doi.org/10. 1002/inf2.12148

6. Bendersky V. A., Leggett A. J., OvchinnikovYu. N., Krevchik V. D. [et al.]. Controlleddissipativetunneling. Tunnel transport in low-dimensional systems : monograph. M., 2011–2012. 496 p.

7. Dahnovsky Yu., Leggett A. J., Semenov M. B., Krevchik V. D. [et al.]. Transfer processes in low-dimensional systems. Tokyo : UT Research Institute Press, 2005. 690 p.

Информация об авторах

Кревчик Владимир Дмитриевич, доктор физико-математических наук, профессор, заведующий кафедрой «Физика», декан факультета «Информационные технологии и электроника», Пензенский государственный университет.

Семёнов Михаил Борисович, доктор физико-математических наук, профессор, профессор-консультант кафедры «Физика», Пензенский государственный университет.

Кревчик Павел Владимирович, магистрант, Пензенский государственный университет.

Авторы заявляют об отсутствии конфликта интересов.