ВЕСТНИК
НОВОСИБИРСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

Научный журнал
Основан в ноябре 1999 года

Серия: Информационные технологии 2020. Том 18, № 2

СОДЕРЖАНИЕ

ISSN 1818-7900 (Print). ISSN 2410-0420 (Online)
Вестник НГУ. Серия: Информационные технологии. 2020. Том 18, № 2
Vestnik NSU. Series: Information Technologies, 2020, vol. 18, no. 2

ISSN 1818-7900 (Print). ISSN 2410-0420 (Online)
Вестник НГУ. Серия: Информационные технологии. 2020. Том 18, № 2
Vestnik NSU. Series: Information Technologies, 2020, vol. 18, no. 2

V E S T N I K
NOVOSIBIRSK STATE UNIVERSITY

Scientific Journal
Since 1999, November
In Russian

Series: Information Technologies 2020. Volume 18, № 2

CONTENTS

ISSN 1818-7900 (Print). ISSN 2410-0420 (Online)
Вестник НГУ. Серия: Информационные технологии. 2020. Том 18, № 2
Vestnik NSU. Series: Information Technologies, 2020, vol. 18, no. 2

The journal is published quarterly in Russian since 1999
by Novosibirsk State University Press

The address for correspondence

Faculty of Information Technologies, Novosibirsk State University

1 Pirogov Street, Novosibirsk, 630090, Russia

Tel. +7 (383) 363 42 46

E-mail address: inftech@vestnik.nsu.ru

On-line version: http://elibrary.ru

ISSN 1818-7900 (Print). ISSN 2410-0420 (Online)
Вестник НГУ. Серия: Информационные технологии. 2020. Том 18, № 2
Vestnik NSU. Series: Information Technologies, 2020, vol. 18, no. 2

УДК 004.4
DOI 10.25205/1818-7900-2020-18-2-5-14

Система управления оценочными средствами

К. В. Баталин, Г. Э. Яхъяева

Новосибирский государственный университет
Новосибирск, Россия

Аннотация

В статье описана программная система управления оценочными средствами преподавателя, позволяющая в полуавтоматическом режиме создавать комплект оценочных документов.

В основе алгоритма генерации оценочного документа лежит алгоритм кластеризации категорийных данных. Представлена модификация алгоритма CLOPE, позволяющая автоматически определять необходимое число кластеров в зависимости от входных данных. Также данная модификация решает проблему маленьких кластеров и проблему категорийной кластеризации числовых атрибутов.

Описан итерационный алгоритм формирования комплекта оценочных документов, состоящий из заданного пользователем числа однотипных по структуре, но уникальных по содержанию вариантов оценочных документов.

Ключевые слова

оценочный документ, комплект оценочных документов, кластеризация, категорийные данные, CLOPE

Для цитирования

Баталин К. В., Яхъяева Г. Э. Система управления оценочными средствами // Вестник НГУ. Серия: Информационные технологии. 2020. Т. 18, № 2. С. 5–14. DOI 10.25205/1818-7900-2020-18-2-5-14

Assessment Means Management Software

K. V. Batalin, G. E. Yakhyaeva

Novosibirsk State University
Novosibirsk, Russian Federation

Annotation

The program system of the management of assessment tools for educational process described in the article. The system reduces the time that the teacher spends on the preparation of test materials, allowing you to create a set of assessment documents in a semi-automatic mode.

The system implements the ability to create and manage task banks. Each task in the bank has a set of attributes that are involved in the process of generating assessment documents. The system also implements the ability to automatically generate the required number of variants of the assessment documents. At the same time, the algorithm for generating a set of assessment documents works in such a way that, on the one hand, one set includes the most similar variants of assessment documents in structure, and on the other hand, each assessment document in the set is unique.

The algorithm for generating a set of assessment documents is based on the clustering algorithm for categorical data. In this paper a modification of the CLOPE algorithm was submitted, which allows you to automatically determine the required number of clusters, depending on the input data. Also, this modification solves the problem of small clusters and the problem of categorical clustering of numerical attributes.

The paper also describes an iterative algorithm for the generation of a set of assessment documents.

Keywords

assessment document, a set of assessment documents, clustering, categorical data, CLOPE

For citation

Batalin K. V., Yakhyaeva G. E. Assessment Means Management Software. *Vestnik NSU. Series: Information Technologies*, 2020, vol. 18, no. 2, p. 5–14. (in Russ.) DOI 10.25205/1818-7900-2020-18-2-5-14

Введение

Современное общество с каждым днем овладевает все большим набором знаний. Наука и технологии, а следовательно, и образование развиваются. Образовательные курсы меняют направленность, добавляются новые темы, покрываются новые компетенции, старые подходы адаптируются под современные методы. Все это влечет за собой и изменение способов оценивания и тестирования знаний студентов [1].

Программа учебных курсов регулярно модифицируется, поэтому преподавателям приходится каждый раз не только создавать новые оценочные материалы, но и подготавливать их для проверочных работ: группировать задания по множеству критериев, составлять похожие между собой варианты проверочных работ. Еще одной причиной регулярного составления оценочных материалов является общедоступность старых вариантов проверочных работ. Задания и вопросы публикуются студентами прошлых курсов, что снижает эффективность повторного использования большинства из них.

Составление проверочных материалов – трудоемкое занятие, которое отнимает значительное время у преподавателей, поэтому актуальным становится создание автоматизированных систем, облегчающих работу преподавателя с оценочными средствами.

Обзор существующих инструментальных средств

Все существующие на сегодняшний день системы работы с оценочными материалами по степени автоматизации процесса подготовки оценочных материалов можно разделить на три класса [2]:

1) системы, решающие только проблемы оформления;
2) системы, решающие проблемы генерации банка вопросов;
3) системы, решающие проблемы генерации оценочных материалов на основе банка вопросов.

Рассмотрим каждую группу более подробно.

Системы, решающие проблемы оформления. Системы данного типа позволяют выбирать тип вопроса, вводить сам вопрос и ответ к нему, вводить дополнительную информацию, необходимую для оформления, генерировать файл с оценочными материалами, готовый к печати. В системах данного типа не идет речи об автоматическом разбиении вопросов, не говоря уже о генерации разных вариантов. Основная цель программных систем данного типа – оформление оценочных средств. Примерами систем данного класса являются следующие.

Testmoz [1] – англоязычный онлайн-сервис для создания оценочных материалов. Данный сервис предоставляет наибольшее число возможностей среди программных решений в данной категории. В системе есть поддержка разных типов вопросов, есть возможность проводить тесты онлайн, удобная форма обратной связи со студентами, подготовка документа к печати. Форма обратной связи позволяет преподавателю сообщить студенту об ошибках, дать советы по исправлению или уточнить ответы. В системе есть кастомизация формата печати, можно менять оформление.

EasyTestMaker [2] – онлайн генератор тестов. Есть возможность создавать различные тестовые задания: множественный выбор, заполнение пустых полей, короткий ответ и т. д. Есть функционал перевода оценочного документа в формат .pdf или .doc для дальнейшей распечатки.

Moodle (Modular Object-Oriented Dynamic Learning Environment) [3] – одна из самых популярных систем электронного обучения [3]. Она переведена более чем на 100 языков, и ею

[1] testmoz.com
[2] www.easytestmaker.com
[3] www.moodle.org

пользуются крупные университеты во всем мире. Модуль для проведения тестов (quizzes) в Moodle позволяет создавать тесты как контрольные работы (т. е. определять такие атрибуты, как дата начала, окончания, лимит времени и др.), разрабатывать тестовые задания и вносить их в банк вопросов, наполнять тест заданиями из банка вопросов и определять его внутреннею структуру.

Системы, решающие проблемы генерации банка вопросов. Данный тип систем на вход чаще всего получает текст, по которому проводится тест. На выходе выдает список вопросов для этого текста.

Ярким представителем этой группы программных систем является система Quillionz [4]. Данная система старается автоматизировать весь процесс генерации оценочных материалов начиная составлением банков вопросов, заканчивая подготовкой проверочного материала к печати. Работа Quillionz основана на использовании нейронных сетей. Для работы с Quillionz необходимо загрузить в нее текст, в котором система с помощью искусственного интеллекта найдет ключевые слова, а далее сгенерирует вопросы к ним. Преподаватель может влиять на выбор ключевых слов: убирать предложенные системой ключевые слова, добавлять собственные.

Следующим шагом система создаст банк вопросов к выбранным ключевым словам. Вопросы могут быть по типу “дайте определение”, “выберите верный вариант ответа”, “объясните, почему верно суждение”. Пользователь системы выбирает понравившиеся вопросы, подготавливает документ с выбранными вопросами к печати. Повлиять на формат вопросов, на процесс генерации, на оформление итоговых документов преподаватель не может.

Система работает только с английскими текстами, вопросы получаются однотипными. В системе нет возможности генерировать несколько вариантов оценочных материалов по одному тексту.

Системы, решающие проблемы генерации оценочных материалов на основе банка вопросов. Данный тип систем на вход принимает список вопросов, на выходе выдает разбиение банка вопросов по нескольким вариантам

Генератор ТСПУ [5] создан специально для Томского государственного педагогического университета. Генератор предоставляет онлайн-форму. В форме есть набор заранее зафиксированных полей. Есть вариант оформления билетов для 2-х или 3-х вопросов. Нет возможности указать собственное количество вопросов. Банк вопросов может быть задан как один для всех вопросов, так и отдельный для каждого вопроса. В качестве вопроса выступает текстовая строка. Каждый вопрос располагается на отдельной строке. На выходе система дает готовый для печати документ. Данная система поддерживает только один тип вопросов, строго заданы шаблон и оформление билетов, нет возможности задать собственное количество вопросов в билете, для выбора вопросов используется примитивный алгоритм выбора случайного.

Генератор билетов v2.0 (Ticket Generator) [6] дает возможность не только генерировать проверочные материалы, но и управлять оценочными материалами. Преподаватель может подготавливать и сохранять банки вопросов, просматривать и редактировать их позднее, экспортировать и делиться ими с другими преподавателями. Есть возможность выбора разных шаблонов оформления для будущего проверочного материала: можно выбрать среди заранее подготовленных стандартных шаблонов или создать свой собственный. Есть возможность управления результатами генерации: изменение порядка вопросов, форматирование вопросов, задание разного типа вопросов, выбор категории вопроса. Помимо этого в системе для каждого вопроса можно установить сложность вопроса, объем ответа, оценку за правильный ответ. Но эти данные не будут участвовать в процессе генерации проверочного материала.

[4] www.quillionz.com
[5] http://test.tspu.ru/gentickets
[6] http://kaiu.narod.ru/Genbilet/Genbilet.html

Они хранятся для того, чтобы преподаватель в будущем мог их просматривать и редактировать. Сам процесс генерации примитивный, преподаватель фиксирует категории для каждого вопроса, а далее случайным образом выбираются вопросы для теста.

ExamView [7] – программный комплекс, позволяющий управлять оценочными материалами и генерировать готовые к печати вопросы, имеет большую гибкость в управлении материалами, поддержаны дополнительные функции работы с банками вопросов: слияние, автовыборка на основе критериев. Система имеет удобный пошаговый помощник генерации билетов. На каждом шаге преподавателю объясняется, какие данные необходимо ввести, как дальше эти данные будут использоваться. Помимо управления банками, система позволяет гибко манипулировать оформлением оценочных материалов. Предустановленных шаблонов больше, чем в предыдущем варианте, интерфейс управления форматированием лаконичнее и понятнее.

Система Card_gen [8] представлена в виде макроса Microsoft Word. У данного подхода есть свои преимущества и недостатки. Среда Microsoft Word знакома большинству современных пользователей, поэтому порог вхождения в систему Card_gen низкий, не требуются дополнительные инструкции по работе с макросом. Система не требует дополнительной установки, если у пользователя уже есть предустановленный Microsoft Word. Иначе использование системы может стать проблемой, так как среда Microsoft Word является платным решением. Списки вопросов система берет из другого документа, который находит в той же директории. Поддерживается только один формат документов с вопросами. По умолчанию алгоритм генерации основан на выборе случайного вопроса. Если необходимо поменять оформление оценочного материала или процесс генерации, то нужно делать исправление в исходном коде макроса. В макросе нет интерфейса для работы с генератором, что связано в первую очередь с концепцией макросов Microsoft Word.

Семантическая модель

Семантическая модель предметной области строится на основе четырехслойной модели представления знаний [4; 5]. Кортеж K,σ,T^a,T^s,T^f является формальным представлением семантической модели предметной области, где K – множество прецедентов предметной области, σ – сигнатура, т. е. набор ключевых понятий, T^a – аналитическая теория предметной области, T^s – теория область и T^f нечеткая теория области.

Разрабатываемая программная система относится к классу систем, решающих проблемы генерации оценочных документов на основе банка вопросов. Центральным понятием в разрабатываемой системе является понятие *задания* [6]. Каждое задание состоит из *тела задания* и набора *атрибутов*. Тело задания формализуется в виде предложения (или нескольких предложений) естественного языка, которые хранятся в системе в виде строковой величины и являются неделимым объектом.

Набор атрибутов у разных заданий может быть разным и формируется пользователем системы. Так, например, можно задать атрибут *тип*, который может принимать следующие значения: *вопрос*, *задача*, *тест*, *списочное задание, шаблонное задание* и т. п. Также очень полезным является атрибут *сложность*. Пользователь сам может устанавливать количество уровней сложности заданий, вводя допустимые значения данного атрибута. Аналитическая теория, т. е. набор аксиом, которым должно отвечать понятие задания, описана в работе [6].

С помощью описания соответствующих атрибутов задание можно привязать к учебному плану факультета: к отдельному модулю учебного плана, модулю дисциплин, одной отдельной дисциплине или к отдельной теме дисциплины. Также задание может быть привязано

[7] https://www.turningtechnologies.com/examview
[8] https://github.com/progsan/card_gen

к профессиональному стандарту или к отдельным его профессиональным функциям. Задание может быть привязано к некоторой компетенции или группе компетенций [7].

Банк заданий – это множество заданий и набор их атрибутов. Каждый банк заданий характеризуется своим набором атрибутов, т. е. в одном банке хранятся однотипные задания. У разных банков атрибуты могут отличаться.

Оценочный документ строится из конечного числа заданий. Число заданий в оценочном документе задается пользователем. В процедуре генерации оценочного документа могут участвовать несколько различных банков заданий. Алгоритм выбора заданий для оценочного документа в данной системе не является случайным, он учитывает атрибуты каждого задания.

На первом этапе формирования оценочных документов создается шаблон оценочного документа. Сначала задается количество заданий, входящих в формируемый шаблон. Далее пользователь настраивает шаблон каждого задания, которое он хочет получить в оценочном документе, т. е. задает банк, задания которого необходимо использовать для генерации, и маску задания.

Маска задания – это набор критериев, которым должны удовлетворять атрибуты задания в формируемом оценочном документе. Маска задается в виде строки, формат которой удовлетворяет грамматике

$$S \to S \vee T \,|\, T,\ \ T \to T \wedge M \,|\, M,\ \ M \to a \,|\, S,$$

где *a*– атом в формате

name OPERATION value,

где *name* – название атрибута, *value* – значение для проверки атрибута, *OPERATION* – операция для проверки атрибута и значения.

$$OPERATION \in \ =, \neq, >, \geq, <, \leq \ .$$

В маске могут участвовать только те атрибуты, которые есть в выбранном банке заданий. Пример маски:

Тип = вопрос с открытым ответом, Сложность = легкое задание.

Набор масок задания и список выбранных банков заданий и задают шаблон оценочного документа.

Набор оценочных документов, соответствующих одному шаблону (т. е. вариантов оценочных документов), образует *комплект оценочных документов*. Число вариантов, которое необходимо получить на выходе работы системы, также задается пользователем. При этом алгоритм генерации комплекта оценочных документов работает так, что в один комплект, с одной стороны, входят максимально схожие по структуре варианты оценочных документов, а с другой стороны, каждый оценочный документ в комплекте уникален.

Алгоритм кластеризации категорийных данных

Для каждой маски задания создается *класс заданий*, удовлетворяющих данной маске, т. е. заданий, для которых все критерии маски выполнены. Далее, при формировании оценочного документа, из каждого класса заданий выбирается по одному заданию. Заметим, что одно и то же задание может подходить под разные маски, а значит, классы заданий могут пересекаться. Более того, в шаблоне оценочного документа для разных заданий могут быть заданы одинаковые маски. Следовательно, соответствующие этим маскам классы заданий будут совпадать. При формировании оценочного документа естественным является требование, чтобы оценочный документ не имел одинаковых заданий. Также желательно, чтобы оценоч-

ный документ охватывал как можно больше проверяемого контента, т. е. чтобы задания, входящие в оценочный документ, различались по атрибутам, не входящим в маску. Эта проблема решается при помощи кластеризации каждого класса заданий.

Атрибуты заданий принимают, как правило, значения в шкале наименований, т. е. являются качественными, а не количественными. Следовательно, для кластеризации классов заданий необходимо применять алгоритмы кластеризации категорийных данных. Самым популярным, легко программируемым, быстрым и масштабируемым является алгоритм CLOPE [8; 9]. В данном алгоритме коэффициент отталкивания подбирается автоматически. Алгоритм обладает следующим свойством: чем больше коэффициент отталкивания, тем больше кластеров получится в результате. Для поиска коэффициента мы применяем метод деления отрезка пополам, подбор осуществляем до тех пор, пока не достигнем нужного числа кластеров. Ожидаемым числом кластеров является количество ключей класса в шаблоне оценочного материала с погрешностью 2.

Если применять CLOPE в оригинальном виде, то возникают следующие проблемы [10].

1. Маленькие кластеры, которые получаются после кластеризации. Задания из таких кластеров будут часто повторяться при формировании комплекта оценочных документов.
2. Некоторые атрибуты логичнее и проще задать в виде чисел. Например, размер ответа. Классический алгоритм CLOPE плохо работает с числовыми признаками.

Для решения первой проблемы можно либо удалять, либо объединять маленькие кластеры. В данной предметной области потеря оценочных материалов критична, поэтому был выбран второй вариант – объединение. Для процесса объединения используется функция вычисления стоимости добавления. Стоимость удаления при этом не учитывается. Для разных объемов данных граничное значение маленького кластера может меняться. Поэтому данный критерий вычисляется динамически. Сначала находится медиана по числу объектов во всех кластерах. Границей маленького кластера будет максимум между значением по умолчанию – 2 и $[median \times 75\%] + 1$.

Таким образом удалось достичь адаптивного вычисления критерия маленького кластера в зависимости от результата кластеризации. Слияние кластеров происходит итерационно. Перед началом процесса формируется несколько списков: $S[]$ – список маленьких кластеров, $C[]$ – список кластеров, с которыми происходит объединение. Изначально $C[]$ равен результату кластеризации. Далее запускаем алгоритм удаления маленьких кластеров:

1. Выбираем маленький кластер $S[i]$. Если он уже не удовлетворяет критерию, удаляем его из $S[]$, переходим к следующей итерации.
2. Удаляем выбранный кластер $S[i]$ из $C[]$.
3. Для каждого объекта из $S[i]$ находим наиболее подходящий кластер из $C[]$ с использованием функции стоимости. Переносим объекты в найденные кластеры.
4. Удаляем $S[i]$.

Данный алгоритм удаляет не каждый маленький кластер. Если маленький кластер перестал быть таковым в ходе предыдущих итераций, то он не будет удален.

Для решения второй проблемы необходимо понять, почему CLOPE начинает плохо работать, когда мы используем числовые атрибуты. Дело в том, что для алгоритма даже соседние числа являются абсолютно разными. Из-за этого число итераций сравнений растет, а объекты с похожими атрибутами воспринимаются как разные. Для решения данной проблемы числовые признаки в нашем алгоритме преобразуются в интервалы. Во время вычисления стоимости проверяется не равенство числовых атрибутов, а их принадлежность к одному интервалу. Встает вопрос о том, на какое количество интервалов разбивать атрибут. Для этого используется правило Стерджеса [11]:

$$num = [log_2 N] + 1,$$

где N – количество уникальных значений атрибута.

Алгоритм генерации комплекта оценочных документов

Следующим этапом происходит генерация комплекта оценочных документов на основе результатов работы алгоритма кластеризации. Это итерационный этап. Для каждого задания в шаблоне выбирается случайный вопрос из случайного кластера класса.

На этом этапе возникает проблема дублирования данных между разными классами. Для ее решения выбор заданий из кластеров происходит при необходимости с элементами перебора доступных заданий. Рассмотрим данную проблему подробнее, на примере:

Кластер 1: Задание 1, Задание 2
Кластер 2: Задание 1, Задание 3
Кластер 3: Задание 2, Задание 3

Предположим, что выбор вопросов из кластеров будет происходить в порядке возрастания номера кластера. В кластере 1 выбираем вопрос 2, в кластере 2 выбираем вопрос 3. После этого у нас нет возможности выбрать вопрос из кластера 3, потому что все имеющиеся в нем вопросы уже были использованы на предыдущих шагах. Поэтому происходит изменение выбранного вопроса в предыдущих кластерах, начиная с последнего. Из кластера 2 выбираем вопрос 1, тогда у нас появляется возможность выбрать вопрос 3 из кластера 3. Таким образом, решается обозначенная выше проблема.

Для того чтобы в один оценочный документ не попадали задания из одного кластера, итерации проводятся не только по заданиям, но и во время выбора кластера. В начале итерации фиксируется список доступных кластеров. Среди доступных выбирается случайный. Выбранный кластер убирается из списка доступных кластеров до следующей итерации. Итерация заканчивается, когда заканчивается список доступных кластеров.

Алгоритм генерации комплекта оценочных документов:

1. Итерируем по заданному количеству вариантов оценочных документов. Повторяем алгоритм до тех пор, пока не будет сгенерировано нужно число вариантов.
2. Создаем список результирующих заданий для варианта $Q[]$. Запускаем генерацию варианта.
3. Если номер генерируемого задания в варианте больше, чем требовалось, то выход.
4. K – ключ класса для текущего задания. По ключу получаем следующий доступный набор заданий B для класса K.
5. $BNum$ – количество заданий в наборе B. Обходим в случайном порядке задания из B.
6. Если задание уже есть в $Q[]$, то берем следующее. Запоминаем выбранное задание, кладем его в $Q[i]$.
7. Переходим к выбору задания $i + 1$ на шаг 5.
8. Если на шаге 6 удалось выбрать задание $i + 1$, завершаем выбор задания i.
9. Иначе удаляем задание i из $Q[]$. Переходим к следующему заданию из набора B. Переходим на шаг 5.
10. Если обошли все $BNum$ заданий из набора B, значит, доступных заданий нет: все уже были ранее использованы на предыдущих итерациях. Завершаем шаг с неудачей, возвращаемся к выбору задания $i – 1$.

Данный подход автоматически обрабатывает ситуацию, когда количество кластеров не совпадает с нужным количеством заданий в шаблоне. В итоге получаем необходимое количество вариантов сгенерированных оценочных материалов.

Заключение

В статье описана программная система управления оценочными средствами преподавателя. Система сокращает время, которое преподаватель тратит на подготовку проверочных материалов, позволяя в полуавтоматическом режиме создавать комплект оценочных документов.

В системе реализована возможность создавать и управлять банками заданий. Каждое задание в банке обладает набором атрибутов, которые участвуют в процессе генерации оценочных документов. В системе также реализована возможность автоматически генерировать нужное количество вариантов оценочных документов. При этом алгоритм генерации комплекта оценочных документов работает так, что в один комплект, с одной стороны, входят максимально схожие по структуре варианты оценочных документов, а с другой – каждый оценочный документ в комплекте уникален.

В основе алгоритма генерации комплекта оценочных документов лежит алгоритм кластеризации категорийных данных. В рамках данной работы была разработана модификация алгоритма CLOPE, позволяющая автоматически определять необходимое число кластеров, в зависимости от входных данных. Также данная модификация решает проблему маленьких кластеров и проблему категорийной кластеризации числовых атрибутов.

Список литературы

1. **Долгих М. В.** Формирование фонда оценочных средств как необходимое условие реализации основной профессиональной образовательной программы // Вестник Южно-Уральского профессионального института. 2014. № 1 (13). С. 36–45.
2. **Sharp V. F.** Computer Education for Teachers: Integrating Technology into Classroom Teaching. John Wiley & Sons, 2008, 416 p.
3. **Анисимов А. М.** Работа в системе дистанционного обучения Moodle: Учеб. пособие. 2-е изд. Харьков: ХНАГХ, 2009. 292 с.
4. **Найданов Ч. А., Пальчунов Д. Е., Сазонова П. А.** Теоретико-модельные методы интеграции знаний, извлечённых из медицинских документов // Вестник НГУ. Серия: Информационные технологии. 2015. Т. 13, № 3, С. 29–41.
5. **Palchunov D., Yakhyaeva G., Dolgusheva E.** Conceptual Methods for Identifying Needs of Mobile Network Subscribers. In: CEUR Workshop Proceedings, vol. 1624, p. 147–160.
6. **Яхъяева Г. Э., Абсайдульева А. Р.** Семантический подход к моделированию фонда оценочных средств // Вестник НГУ. Серия: Информационные технологии. 2018. Т. 16, № 2. С. 113–121.
7. **Баталин К. В., Мамеев Н. С., Попова К. Ю., Рыжаков И. Д., Яхъяева Г. Э.** Программная система управления образовательным процессом ИТОС // Вестник НГУ. Серия: Информационные технологии. 2018. Т. 16, № 4. С. 20–30.
8. **Yang Y., Guan H., You J.** CLOPE: A fast and Effective Clustering Algorithm for Transactional Data In: Proc. of SIGKDD'02. Edmonton, Alberta, Canada, 2002.
9. **Паклин Н. Б.** Кластеризация категорийных данных: масштабируемый алгоритм CLOPE. Научная библиотека BaseGroup Labs. URL: http://www.basegroup.ru/library/analysis/clusterization/clope.
10. **Баталин К. В.** Оптимизация метода кластеризации категорийных данных базы оценочных материалов // Материалы Всерос. конференции с международным участием «Знания – Онтологии – Теории» (ЗОНТ-2019). Новосибирск, 2019. С. 414.
11. **Шорохов И. С., Кисляк Н. В., Мариев О. С.** Статистические методы анализа: Учеб. пособие. Екатеринбург: Изд-во Урал. ун-та, 2015. 300 с.

References

1. **Dolgikh M. V.** Formation of the Fund of evaluation funds as a necessary condition for the implementation of the main professional educational program. *Bulletin of the South Ural professional Institute*, 2014, no. 1 (13), p. 36–45. (in Russ.)
2. **Sharp V. F.** Computer Education for Teachers: Integrating Technology into Classroom Teaching. John Wiley & Sons, 2008, 416 p.
3. **Anisimov M. A.** Work in system of distance learning Moodle. Textbook. 2nd ed. Kharkov, KhNAGKh Press, 2009, 292 p. (in Russ.)
4. **Naidanow C. A., Palchunov D. E., Sazonov P. A.** Model-Theoretical methods for integration of knowledge extracted from medical records. *Vestnik NSU. Series: Information Technologies*, vol. 13, no. 3, 2015, p. 29–41. (in Russ.)
5. **Palchunov D., Yakhyaeva G., Dolgusheva E.** Conceptual Methods for Identifying Needs of Mobile Network Subscribers. In: CEUR Workshop Proceedings, vol. 1624, p. 147–160.
6. **Yakhyayeva G. E., Absaidulieva A. R.** Semantic approach to the modeling of the Foundation assessment tools. *Vestnik NSU. Series: Information Technologies*, 2018, vol. 16, no. 2, p. 113–121. (in Russ.)
7. **Batalin K. V., Mameev N. S., Popova K. Yu., Ryzhakov I. D., Yakhyaeva G. E.** Software system of educational process management ITOS. *Vestnik NSU. Series: Information Technologies*, 2018, vol. 16, no. 4, p. 20–30. (in Russ.)
8. **Yang Y., Guan H., You J.** CLOPE: A fast and Effective Clustering Algorithm for Transactional Data In: Proc. of SIGKDD'02. Edmonton, Alberta, Canada, 2002.
9. **Paklin N. B.** Clusterization of categorical data: a scalable SLOPE algorithm. BaseGroup Labs scientific library. URL: http://www.basegroup.ru/library/analysis/clusterization/clope. (in Russ.)
10. **Batalin K. V.** Optimization of the clustering method of categorical data of the evaluation materials database. In: Proceedings of the All-Russian conference with international participation "Knowledge – Ontology – Theory" (UMBRELLA-2019). Novosibirsk, 2019, p. 414. (in Russ.)
11. **Shorokhov I. S., Kislyak N. V., Mariev O. S.** Statistical methods of analysis. Studies guide. Ekanerinburg, Ural State Uni. Press, 2015, 300 p. (in Russ.)

Материал поступил в редколлегию
Received
10.05.2020

Сведения об авторах Information about the Authors

Баталин Кирилл Вячеславович, магистрант ФИТ Новосибирского государственного университета (Новосибирск, Россия)
k.v.batalin@gmail.com

Яхъяева Гульнара Эркиновна, кандидат физико-математических наук, доцент, доцент кафедры общей информатики ФИТ Новосибирского государственного университета (Новосибирск, Россия)
gul_nara@mail.ru

Information about the Authors

Kirill V. Batalin, Master of Science, Novosibirsk State University (Novosibirsk, Russian Federation)
k.v.batalin@gmail.com

Gulnara E. Yakhyaeva, Associate Professor of the Department of Information Technology, Novosibirsk State University (Novosibirsk, Russian Federation)
gul_nara@mail.ru

УДК 004.4'422
DOI 10.25205/1818-7900-2020-18-2-15-30

Разработка внутреннего представления компилятора Kotlin / Native и оптимизаций на его основе

С. С. Боголепов

Новосибирский государственный университет
Новосибирск, Россия

Аннотация

Котлин – это статически типизированный язык программирования, который поддерживает объекно-ориентированную и функциональную парадигмы программирования. Изначальной целевой платформой была выбрана JVM, однако затем была добавлена возможность транспиляции в JavaScript и компиляции под нативные платформы с помощью LLVM (Kotlin / Native). Первые две платформы представляют собой хорошо развитые виртуальные машины, способные выполнять продвинутую оптимизацию программ во время исполнения. Однако в случае нативных платформ оптимизацию необходимо выполнять во время компиляции.

На данный момент в Kotlin / Native отсутствуют многие оптимизации, из-за чего производительность порождаемого кода во многих случаях получается низкой. В этой работе описан способ решения данной проблемы с помощью введения дополнительного внутреннего представления, основанного на SSA-форме, и реализации escape-анализа на его основе. Результаты экспериментов показали, что этот подход способен значительно улучшить производительность.

Ключевые слова

Kotlin, static single assignment, escape-анализ, оптимизация, LLVM

Для цитирования

Боголепов С. С. Разработка внутреннего представления компилятора Kotlin / Native и оптимизаций на его основе // Вестник НГУ. Серия: Информационные технологии. 2020. Т. 18, № 2. С. 15–30. DOI 10.25205/1818-7900-2020-18-2-15-30

Development of Kotlin / Native Intermediate Representation and Optimizations

S. S. Bogolepov

Novosibirsk State University
Novosibirsk, Russian Federation

Abstract

Kotlin is a statically typed programming language that supports object-oriented and functional programming. It supports JVM, JS and native platforms via LLVM (Kotlin / Native). The first two targets are backed with well-developed virtual machines that can perform advanced program optimizations at runtime. However, for native platforms, all optimizations must be performed at compile time.

Currently Kotlin / Native lacks many optimizations, which is why the performance of the generated code is poor in many cases. This paper describes a way to solve this problem by introducing an additional SSA-based intermediate representation and implementing escape analysis using it. Experimental results have shown that this approach can significantly improve performance.

Keywords

Kotlin, static single assignment, escape analysis, optimization, LLVM

For citation

Bogolepov S. S. Development of Kotlin / Native Intermediate Representation and Optimizations. *Vestnik NSU. Series: Information Technologies*, 2020, vol. 18, no. 2, p. 15–30. (in Russ.) DOI 10.25205/1818-7900-2020-18-2-15-30

Введение

Kotlin / Native – это AOT (Ahead-of-Time) компилятор языка программирования Котлин. Для компиляции в машинный код используется LLVM. Поддерживается большое количество целевых платформ, таких как iOS, macOS, Linux (arm64, x64, mips32) и многие другие.

Примечательной особенностью Kotlin / Native является то, что он компилируется в режиме «закрытого мира», что отличает его от большинства других промышленных компиляторов. Режим «закрытого мира» означает, что во время компиляции доступен весь граф зависимостей текущей единицы трансляции, что позволяет делать глобальные анализ и оптимизацию программы. Например, известна вся иерархия классов, что упрощает оптимизацию виртуальных вызовов.

Недостатком по сравнению с «открытым миром» является усложнение раздельной компиляции, а также трудности с одновременным использованием нескольких динамических библиотек, написанных на Kotlin / Native. Например, это приводит к тому, что стандартная библиотека и другие зависимости включены в каждый бинарный файл. Если одна зависимость используется в двух разных бинарных файлах, то это разные с точки зрения идентичности декларации, так как они принадлежат к разным «мирам».

Внутреннее представление компилятора Kotlin / Native

Внутреннее (или промежуточное) представление (Intermediate Representation, IR) компилятора – структура данных, используемая компилятором для представления исходного кода входной программы. Многие компиляторы используют несколько внутренних представлений на разных этапах компиляции, так как некоторые алгоритмы трансляции и оптимизации удобнее реализовывать с помощью одного представления, а другие – с помощью другого.

Рассмотрим, как эволюционировал компилятор Котлина с точки зрения используемых внутренних представлений.

Program Structure Interface. Изначально в компиляторе Котлина не было продвинутого внутреннего представления, пригодного для оптимизаций. PSI (Program Structure Interface, представление кода, использующееся в Intellij Platform) напрямую транслировался в байткод виртуальной машины Java. Такое же решение использовалось и при трансляции в JavaScript. Данный подход был приемлем, так как в обоих случаях задача оптимизации кода программы целиком ложилась на плечи виртуальной машины. Тем не менее даже для таких примитивных целей (с точки зрения продвинутых компиляторов) это не самый удачный выбор, так как, например, подстановку инлайн-функций приходится выполнять на байткоде JVM, что является крайне нетривиальной задачей. Поэтому вместе с разработкой Kotlin / Native началась разработка полноценного нового внутреннего представления, общего для всех компиляторов.

Древовидное представление. Выбор пал на древовидное представление программ. Оно низкоуровневое, по сравнению с PSI, изменяемое и сериализуемое, что позволяет использовать его для реализации оптимизаций и в качестве формата для дистрибуции библиотек. В отличие от многих других компиляторов, линковка в компиляторе Kotlin / Native происходит на уровне промежуточного представления (а не бинарного кода), поэтому на вход оптимизирующим фазам компилятора подается вся программа с ее зависимостями. Разумеется, это дорогой с точки зрения времени компиляции подход, поэтому для тех режимов компиля-

ции, в которых не важна производительность порождаемого кода, существует кэширование зависимостей в виде бинарного кода.

Над древовидным представлением можно производить понижающие преобразования, что позволяет оптимизировать программный код и упрощает дальнейшую трансляцию. Но так как отсутствует явный граф потока управления, реализация многих оптимизаций, основанных на нем, затруднена. Поэтому, например, оптимизация циклов выполнена ad hoc для известных коллекций из стандартной библиотеки, что существенно ограничивает область ее применения.

Рассмотрим пример. Пусть на вход компилятору дана следующая функция f:

```
@SSA
fun f(x: Boolean): Int {
    val a = if (x) {
        A(5)
    } else {
        A(6)
    }
    a.print()
    return 0
}
```

Тогда текстовое описание ее древовидного представления выглядит так:

```
FUN name:f visibility:public modality:FINAL <>
(x:kotlin.Boolean) returnType:kotlin.Int
      annotations:
        SSA
      VALUE_PARAMETER name:x index:0 type:kotlin.Boolean
      BLOCK_BODY
        VAR name:a type:<root>.A [val]
          WHEN type=<root>.A origin=IF
            BRANCH
              if: GET_VAR 'x: kotlin.Boolean'
type=kotlin.Boolean
              then: BLOCK type=<root>.A
                CONSTRUCTOR_CALL 'public constructor <init>
(arg: kotlin.Int) [primary]' type=<root>.A
                  arg: CONST Int type=kotlin.Int value=5
            BRANCH
              if: CONST Boolean type=kotlin.Boolean value=true
              then: BLOCK type=<root>.A origin=null
                CONSTRUCTOR_CALL 'public constructor <init>
(arg: kotlin.Int) [primary]' type=<root>.A
                  arg: CONST Int type=kotlin.Int value=6
        CALL 'public final fun print (): kotlin.Unit'
type=kotlin.Unit
          $this: GET_VAR 'val a: <root>.A [val]' type=<root>.A
        RETURN type=kotlin.Nothing
          CONST Int type=kotlin.Int value=0
```

LLVM

LLVM (Low-Level Virtual Machine) – это инфраструктура разработки компиляторов [1]. В ее основе лежит низкоуровневое внутреннее представление, основанное на графе потока управления в SSA-форме – LLVM IR (сериализованная форма которого называется «биткод»). Фронтенду компилятора достаточно транслировать свое представление (например, древовидное) в LLVM IR, после чего LLVM выполнит его оптимизацию и компиляцию под целевую платформу. Оптимизации LLVM не привязаны к какому-то конкретному языку и поэтому достаточно низкоуровневые. Пример оптимизаций, которые поддерживает LLVM:

- mem2reg – замена операций работы с памятью на операции работы с виртуальными регистрами;
- Global Value Numbering – поиск общих подвыражений;
- удаление мертвого кода (как целых функций, так и инструкций и базовых блоков).

LLVM лег в основу большого количества современных компиляторов: Clang, Rust, Swift, Julia, Crystal, Scala Native и многих других.

Static Single Assignment (SSA) форма

Промежуточное представление программы, в котором каждое значение присваивается ровно один раз. Данное представление значительно упрощает написание многих видов оптимизаций, так как отсутствуют изменяемые переменные. Важной особенностью данного представления является φ-функция, которая служит для слияния нескольких значений в одно в случае нескольких предшествующих ветвей управления (рис. 1, 2).

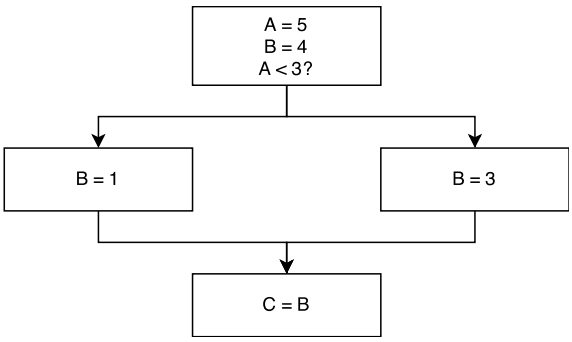

Рис. 1. Пример графа потока управления не в SSA-форме
Fig. 1. Non-SSA Control Flow Graph

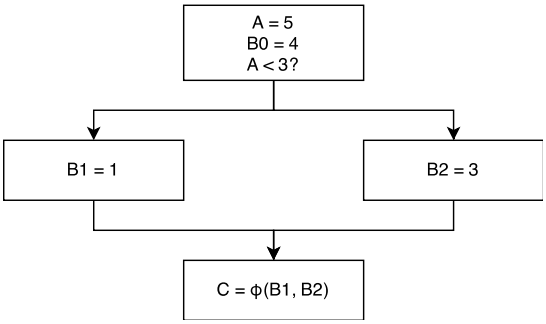

Рис. 2. Пример того же графа управления в SSA-форме
Fig. 2. The same Control Flow Graph in SSA form

Управление памятью

В отличие от большинства реализаций JVM и виртуальных машин JS, в которых для сборки мусора применяется трассирующий коллектор, в Kotlin / Native используется автоматический подсчет ссылок (рис. 3). Наивный алгоритм подсчета ссылок плохо применим к языку Котлин, так как он приводит к утечкам памяти, поэтому используется модификация алгоритма, описанного в работе [2].

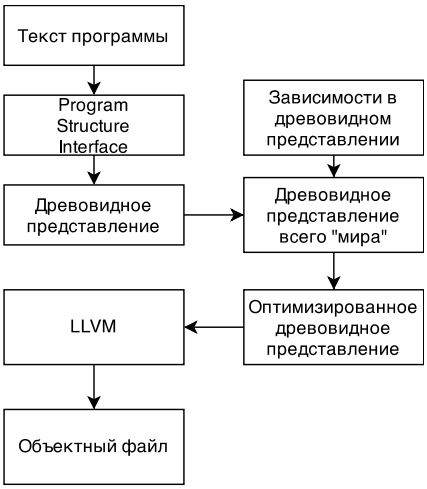

Рис. 3. Текущий пайплайн компилятора Kotlin / Native
Fig. 3. Current Kotlin / Native compiler pipeline

Одной из основных причин для выбора данного алгоритма была необходимость взаимодействия со средой исполнения Objective-C, где также используется подсчет ссылок.

Замеры текущей производительности

Рассмотрим сравнение производительности разных языков [1]. Оно нас интересует из-за того, что в тестовой программе много аллокаций объектов и мало виртуальных вызовов, что позволяет оценить влияние управления памятью на общую производительность кода. Рассмотрим результаты измерения профиля программы с помощью утилиты perf, приведенные в (табл. 1). Как видно из таблицы, все указанные функции (за исключением splitBinary и merge) относятся к управлению памятью.

Данная компонента среды исполнения является бутылочным горлышком Kotlin / Native в плане производительности, так как стиль программирования на языке Котлин предполагает большое количество объектов и присвоения ссылок, что приводит к большому количеству вызовов функций управления памятью. В отличие от трассирующего коллектора, который используется в подавляющем большинстве сред исполнения с автоматической сборкой мусора, в средах исполнения с подсчетом ссылок каждое создание объекта и изменение ссылки приводит к замедлению программы.

[1] https://github.com/frol/completely-unscientific-benchmarks

Таблица 1

Профиль в Unscientific Benchmark

Table 1

Unscientific Benchmark profile

Функция и ее происхождение	Доля процессорного времени
garbageCollect (управление памятью)	19,7
allocInstance (управление памятью)	16
kfun:splitBinary (пользовательский код)	10,1
freeContainer (управление памятью)	10
updateHeapRef (управление памятью)	6,3
kfun:merge (пользовательский код)	5,5
ReleaseHeapRefStrict (управление памятью)	5,3
Остальные функции	27,1

Кроме того, ошибочно воспринимать LLVM как JVM, CLR и другие виртуальные машины. Несмотря на то, что фундаментально данные технологии похожи (трансляция и оптимизация некоторого низкоуровневого входного языка в бинарный код), концептуально они совершенно разные: оптимизирующий пайплайн LLVM ожидает на вход достаточно примитивный язык и не умеет оптимизировать паттерны, соответствующие коду, написанному на объектно-ориентированном (ОО) языке, в то время как, например, JVM изначально создавалась с расчетом, что ей на вход подается байткод, который получен трансляцией программы на языке Java практически без оптимизаций.

Возможные решения

Так как нас интересует уменьшение доли вышеперечисленных функций в профиле программы, то возможны два ортогональных подхода:

- динамический – ускорение исполнения путем ручной оптимизации данных функций;
- статический – уменьшение количества вызовов к ним путем компиляторных оптимизаций.

Первый подход более простой, так как не требует внесения изменений в код компилятора, но является ограниченным, поскольку вызовы функций среды исполнения все равно остаются. Кроме того, он никак не улучшает код, порождаемый компилятором. Второй подход потенциально позволяет избавиться вообще от всех операций инкремента / декремента счетчика ссылок в некоторых случаях, но при этом сложнее в реализации.

Для реализации второго подхода часто (в частности, в JVM) используется алгоритм escape-анализа. Цель данного анализа – определить область видимости объекта. Например, если он виден только внутри функции, то его можно аллоцировать на стеке, а если он присваивается в глобальную переменную, то это будет некорректно, и его нужно аллоцировать на куче.

Как и с любым алгоритмом, для реализации escape-анализа очень важно правильное представление входных данных. В данном случае таковым является внутреннее представление компилируемой программы. Рассмотрим, почему ни одно из двух существующих представлений компилятора Kotlin / Native (древовидное и LLVM IR) не являются оптимальными для реализации escape-анализа.

На древовидном IR такую оптимизацию полноценно реализовать оказывается затруднительно, так как отсутствуют явное представление потока управления и операции подсчета

ссылок. В Kotlin / Native на данном представлении реализован алгоритм локального escape-анализа, который крайне консервативен (поддерживает только локальные массивы примитивных типов фиксированной длины).

В LLVM существует развитая инфраструктура для написания анализов и трансформаций, поэтому возникает идея переиспользовать ее для написания собственных. Однако это довольно нетривиально, так как в LLVM отсутствует любая информация о статических типах объектов (например, информация о наследовании), которая необходима для написания высокоуровневых оптимизаций. Ее можно передать в виде метаинформации, но работать с ней в таком случае крайне затруднительно.

Таким образом, возникает идея для добавления еще одного внутреннего представления, которое, с одной стороны, будет знать об особенностях входного языка, а с другой – отражать поток управления и операции подсчета ссылок. Похожие решения применяются во многих других компиляторах, которые используют LLVM (например, Swift, Rust и Scala Native).

В дальнейшем новое внутреннее представление будет именоваться SSA IR (Static-Single Assignment Intermediate Representation).

Описание реализации

SSA IR – это внутреннее представление, основанное на графе потока управления, находящегося в SSA-форме. Рассмотрим основные составляющие данного представления.

Модуль. Это коллекция определений функций и глобальных переменных. На данный момент модуль выполняет единственную роль: это удобный способ запустить какую-то трансформацию или анализ над всеми элементами SSA IR.

Функция. Это именованный направленный граф базовых блоков, соединенных между собой параметризованными ребрами. Параметры функции совпадают с параметрами входного базового блока. У методов классов и интерфейсов нет неявного параметра this, он указывается явно первым параметром.

Для каждой функции хранится ссылка на соответствующую декларацию древовидного представления, из которой можно получить дополнительную информацию (например, область видимости функции).

Базовый блок. Это последовательность инструкций, у которой одна точка входа (первая инструкция) и одна точка выхода (терминальная инструкция). Одной из особенностей реализации является то, что вместо классических φ-функций используются параметры у базовых блоков. Данный подход, во-первых, делает взаимосвязи между базовыми блоками более очевидными, а во-вторых, решает ряд проблем классического подхода. Например:

- φ-функции всегда должны быть в начале базового блока, что усложняет оптимизации;
- невозможность провести два ребра от одного блока к другому.

Кроме того, базовые блоки с параметрами фактически стирают границу между SSA-представлением и Continuation Passing Style [3], что упрощает процесс решения проблем при использовании подобного представления.

Инструкция. Аналогично LLVM IR инструкции представлены в виде трехадресного кода (т. е. имеют явно указанное возвращаемое значение и параметры).

В отличие от LLVM IR, набор инструкций SSA IR позволяет описывать более высокоуровневые операции. Рассмотрим несколько примеров.

- SSAInterfaceCall. Вызов интерфейсного метода.
- SSAVirtualCall. Вызов виртуального метода.
- SSAIncRef/SSADecRef. Инкремент / декремент счетчика ссылок у переданного объекта.
- SSAAlloc. Абстрактная аллокация (на куче или на стеке).

Для того чтобы упростить работу, была добавлена аннотация @SSA, которая означает, что над аннотированной функцией необходимо проводить нижеописанные операции. Таким образом, стало возможным разрабатывать SSA IR постепенно, аннотируя только те функции, конструкции которых поддержаны в представлении (рис. 4).

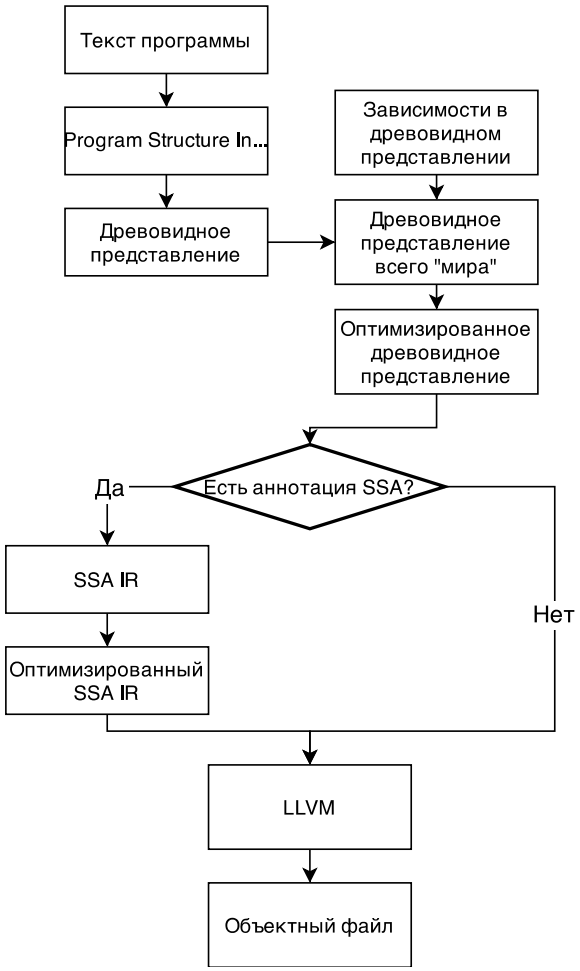

Рис. 4. Новый пайплайн компилятора
Fig. 4. New compiler pipeline

Трансляция в SSA IR

Алгоритм трансляции древовидного представления в SSA IR основан на методе, описанном в работе [4]. В отличие от классического алгоритма, описанного в [5], в данном алгоритме не требуется предварительно строить граф потока управления и фронт доминаторов, что значительно упрощает трансляцию напрямую из древовидного представления.

Алгоритм трансляции

Инструкции древовидного представления, которые выполняются последовательно, оказываются в одном базовом блоке. Когда встречается запись в переменную, мы записываем промежуточное представление, являющееся значением переменной, как *текущее значение пере-*

менной. Соответственно, когда происходит чтение значения переменной, мы его используем. Данный процесс называется *local value numbering.* Целиком заполненный базовый блок помечается как *заполненный.*

Если базовый блок не содержит определения переменной, то мы рекурсивно ищем его среди предков блока. Если предков несколько, то на ребра каждого из них добавляется значение переменной, а базовому блоку добавляется параметр, формальное значение которого указывается в качестве значения переменной. Как можно заметить, при использовании такой нотации (вместо φ-функции) возникает ассоциация с вызовом функции, что значительно упрощает ментальную модель данного представления.

В случае циклов внутри функции может возникнуть рекурсивное определение значения переменной. Поэтому сначала добавляется формальный параметр базового блока, и если при поиске значения переменной мы встречаем формальный параметр, то это означает конец рекурсии.

Поиск значения осуществляется только внутри *заполненных* блоков, так как в незаполненный блок может добавиться новое значение, которое перезапишет предшествующее.

Назовем базовый блок *запечатанным*, если известны все его предки. Стоит отметить, что свойство *запечатанности* не влечет за собой *заполненность* (тривиальный пример – входной базовый блок).

Проблема возникает, когда мы пытаемся найти определение в *незапечатанном* блоке, в котором нет определения этой переменной. В таком случае мы создаем дополнительный параметр блока и указываем его в качестве значения. Для каждого блока мы храним список таких параметров. В дальнейшем, когда происходит *запечатывание* базового блока, для каждого такого параметра осуществляется поиск по алгоритму, описанному ранее.

Полезным побочным эффектом данного алгоритма является то, что ряд оптимизаций происходит прямо во время построения представления. Например, протяжка констант.

Ранее рассмотренная функция f в SSA IR принимает следующий вид:

```
f(%0: bool): (bool) -> int
block entry0():
  condbr %0: bool when_body1() when_cond2()

block when_body1():
  %2 A = allocate
  %3 A = call_direct A.<init> %2: A, 5: int
  br when_exit3(%2: A)

block when_cond2():
  %5 A = allocate
  %6 A = call_direct A.<init> %5: A, 6: int
  br when_exit3(%5: A)

block when_exit3(%8: A):
  %9 type_unk = call_direct A.print %8: A
  br return_block4(0: int)

block return_block4(%11: int):
  ret %11: int
```

Алгоритмы над SSA IR

Все алгоритмы выполнены в форме проходов над представлением, которые запускаются последовательно.

Валидация проверяет инварианты, общие для всех промежуточных состояний представления, а именно:

- все базовые блоки замкнуты;
- все базовые блоки заканчиваются терминальными инструкциями;
- все использования некоторого значения указаны в списке пользователей данного значения;
- инструкция находится внутри базового блока, который указан как ее владелец;
- количество и тип параметров базового блока соответствует количеству и типу аргументов, переданных в него.

Удаление недостижимых блоков. В результате трансляции из древовидного представления или в ходе какого-то преобразования может возникнуть базовый блок, у которого нет предков. Поток управления никак не может прийти в такой базовый блок, поэтому он может быть удален. Данный алгоритм реализован в LLVM, поэтому он не является обязательным. Тем не менее эта оптимизация уменьшает время работы дальнейших алгоритмов и упрощает отладку.

Открытая подстановка геттеров и сеттеров. В Котлине вместо методов вида *getField*() и *setField*() используются *свойства* (property), для которых автоматически генерируются соответствующие методы доступа к полям объекта: геттеры и сеттеры. Их можно переопределять, усложняя поведение, но в большинстве случаев в них нет никакой произвольной логики, кроме обращения к подлежащему полю. Возможность «увидеть» доступ к полям объекта крайне важен для escape-анализа, так как если доступ к полю скрыт внутри метода, то escape-анализ сможет увидеть, что объект *убегает*, через присваивание в поле только на этапе межпроцедурного анализа. Поэтому необходима трансформация, которая осуществляет открытую подстановку (инлайн) таких методов.

Полноценная реализация такой оптимизации является крайне трудоемкой задачей с огромным числом эвристик. Для наших же целей достаточно реализовать простой алгоритм, который подставляет только геттеры и сеттеры с примитивными телами. Компиляция в режиме «закрытого мира» позволяет проанализировать тело вызываемого метода и принять решение о подстановке.

Построение графа вызова функций позволяет выделить компоненты сильной связности (т. е. взаимную рекурсию) и понять, какие функции «скрываются» за виртуальными вызовами. Без него крайне затруднительно проводить межпроцедурные оптимизации.

В компиляторах ООП языков наиболее распространены два алгоритма: Class Hierarchy Analysis и Rapid Type Analysis [6]. В данной работе использована упрощенная версия последнего алгоритма.

Рассмотрим пример работы алгоритма. Пусть на вход компилятору подается следующая программа:

```
interface I {
    fun virtualFn()
}

class A : I {
    override fun virtualFn() {}
}

class B : I {
    override fun virtualFn() {}
}
```

```
class C : I {
    override fun virtualFn() {}
}

@SSA
fun f(isA: Boolean): I = if (isA) A() else B()

@SSA
fun main() {
    f(true).virtualFn()
}
```

В этой программе объект класса C никогда не создается, поэтому в графе вызовов C.virtualFn отсутствует (рис. 5).

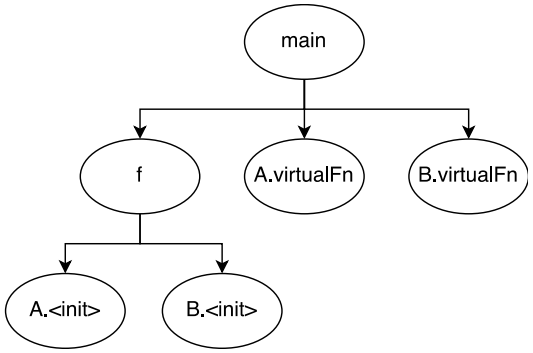

Рис. 5. Пример графа вызовов
Fig. 5. Call graph example

Escape-анализ

Чтобы уменьшить нагрузку на сборщик мусора, необходимо статически вывести время жизни объектов. Те объекты, время жизни которых не превышает время жизни стекового фрейма, можно аллоцировать на стеке. Алгоритм, который реализует такой анализ, называется «escape-анализ». Существует несколько реализаций такого алгоритма. Большинство из них было написано для JVM, которая JIT-компилирует код в режиме открытого мира. Это требует ряда компромиссов, так как в данном случае компилятор разделяет ресурсы компьютера с исполняемым кодом. В нашем же случае компиляция происходит AOT в режиме закрытого мира, что позволяет получить более точные результаты анализа ценой времени компиляции.

Описание работы использованного алгоритма

В данной работе был реализован алгоритм, описанный в статье [7]. Как и многие глобальные анализы, выбранный алгоритм разбивается на две фазы: внутрипроцедурного и межпроцедурного анализа.

Внутрипроцедурный анализ. Эта часть анализа заключается в построении локального графа связей для всех анализируемых функций.

Граф связей. Структура данных, которая используется для вычисления и хранения информации о том, на какие объекты могут ссылаться переменные, поля объектов и параметры функций. Граф связей – это направленный граф (N, E).

N – это множество вершин, которое разделяется на два класса. N_o – множество *объектных* вершин, представляющих объекты. N_r – множество *ссылочных* вершин, представляющих ссылки на объекты. Существует несколько видов ссылочных вершин:

- *локальные* представляют локальные переменные, ссылающиеся на объекты.
- *актуальные* (actual) – параметры метода, аргументы и возвращаемые значения.
- *поля* – поля ссылочного типа.
- *глобальные* – глобальные переменные.

Стоит отметить, что одна объектная вершина может представлять множество вершин времени исполнения (например, если аллокация происходит в цикле).

E – это множество ребер нескольких видов:

- *указатель* – ребро от вершины-ссылки к вершине-объекту;
- *поле объекта* – ребро от вершины-объекта *o* к вершине-ссылке *f*. Существует, только если *f* представляет поле объекта *o*;
- *отложенное* ребро – ребро между двумя ссылками, которое возникает в случае присваивания.

Такие ребра можно удалять с помощью операции *ByPass(q)*. Она заключается в «перекидывании» отложенных ребер, указывающих на *q*, на вершины, на которые указывает *q* (рис. 6).

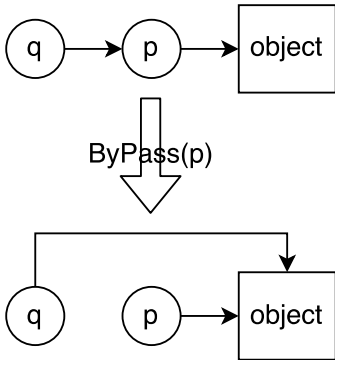

Рис. 6. Пример операции *ByPass(p)*
Fig. 6. ByPass(p) sample

С каждой вершиной ассоциируется одно из двух состояний: *не убегает* или *убегает*. Последнее состояние назначается вершинам, которые живут дольше, чем стековый фрейм анализируемого метода. Состояние *убегает* «растекается» по ребрам, и, таким образом, все транзитивное замыкание «убегающей» вершины становится «убегающим».

Построение графа связей. Граф связей строится последовательным обходом графа потока управления. Циклы обрабатываются до тех пор, пока алгоритм не сойдется. Если спустя фиксированное число итераций алгоритм не сходится, всем вершинам метода консервативно выставляется состояние *убегает*. Граф связей на входе в базовый блок – это объединение всех графов связей его предшественников.

Перед построением графа связей для входного базового блока нужно обработать параметры функции в виде $f = a$, где f – это локальная вершина ссылочного типа, a – актуальная вершина ссылочного типа, которая представляет собой фактический аргумент. О том, как обрабатываются присваивание, рассказано ниже.

Обработка инструкций в базовом блоке.

- Присваивание объекта *O* в вершину *v*. Создается новая объектная вершина для *O*. Затем вызывается *ByPass(v)*, после чего добавляется ребро-указатель от *v* к *O*.

- Присваивание переменной *p* значения переменной *q*. Применяем *ByPass(p)*, затем добавляем *отложенное* ребро от *p* в *q*.
- *p.f* = *q*. Если в графе нет объектных вершин, достижимых из *p* через *указатели* или *отложенные* ребра (например, это параметр функции), то создаем фантомный объект *O* и добавляем ребро-указатель. Потом убеждаемся, что для всех таких объектных вершин существует вершина для поля *f*. Затем для всех *f* добавляем отложенное ребро к *q*. Если *p.f* – это глобальная переменная, то у *q* устанавливается состояние *убегает*.
- *p* = *q.f*. Сначала делаем *ByPass(p)*. Если у *q* нет объектов, на которые она указывает, создаем фантомный объект и добавляем *ребро-указатель*. Создаем ребра и вершины для всех *f*. Затем добавляем *отложенные* ребра от *p* ко всем *f*.

На выходе из метода возвращаемые значения обрабатываются как присваивания в актуальные ссылки. Затем применяется *ByPass* ко всем ссылочным вершинам, чтобы избавиться от всех отложенных ребер. После этого та часть подграфа, которая достижима из *убегающих* вершин формирует *нелокальный подграф*, который представляет собой то, как функция влияет на вызывающую его функцию.

Межпроцедурная часть анализа заключается в уточнении графа вызывающей функции с помощью результатов анализа вызываемой функции.

Аргументы функций рассматриваются как присваивания в *актуальные* ссылочные вершины, т. е. происходит обработка присваивания *a* = *p*, где *a* – это *актуальная ссылка*, которая позднее будет связана с актуальной ссылкой в вызываемой функции (мы ее создаем в самом начале алгоритма).

Сразу после вызова функции мы должны обработать эффект, который вызываемая функция оказывает на вызывающую. Для этого мы должны построить соответствие между двумя графами связи. Мы это делаем начиная с *актуальных* ссылок, которые представляют формальные параметры метода на стороне вызываемой функции с соответствующими ссылками в вызывающей функции. Затем ищем соответствующие объекты в множествах *PointsTo*. Если в вызывающей функции нет соответствующей вершины для объекта, то создается фантомный объект, после чего строим соответствие между полями данных объектов. Этот процесс продолжается рекурсивно до полного замыкания. Далее добавляем соответствующие ребра по аналогичному правилу.

Таким образом, *нелокальный подграф* вызываемой функции копируется в граф вызывающей функции, и его эффект распространяется на последнюю.

Возникает естественный вопрос о рекурсивных вызовах. Для этого выделяются компоненты сильной связности, и межпроцедурный анализ запускается на них до тех пор, пока алгоритм не сойдется.

Расстановка операций управления памятью. Следующая фаза использует результат escape-анализа для расстановки операций инкремента и декремента счетчика ссылок.

В случае если объект не покидает пределов метода, он просто аллоцируется на стеке и помечается специальной маской, чтобы сборщик мусора не удалил такой объект, несмотря на отсутствие живых ссылок на него.

Иначе происходит расстановка операций изменения счетчика ссылок:

- при передаче объекта в качестве аргумента в функцию счетчик увеличивается;
- после возвращения из функции счетчик ссылок уменьшается;
- при записи в поле счетчик ссылок увеличивается, а у объекта, на который до этого указывало поле, уменьшается.

Трансляция из SSA IR в LLVM IR

Так как трансляция в LLVM осуществляется из низкоуровневой вариации SSA IR, то данный процесс становится тривиальным. Параметры базовых блоков транслируются в φ-функции, а большинство остальных инструкций SSA IR – в их аналоги в LLVM. Исключение

составляют операции, которые не отражены в LLVM. Они транслируются в вызовы функций среды исполнения.

Функция f в результате трансляции в LLVM IR выглядит следующим образом. Как видно по инструкции alloca, объект класса A аллоцируется на стеке:

```
define i32 @"kfun:#f(kotlin.Boolean){}kotlin.Int"(i1) #11 {
entry:
  br i1 %0, label %when_body, label %when_cond

when_body:                                        ; preds = %entry
  %1 = alloca %"kclassbody:A#internal"
  %2 = bitcast %"kclassbody:A#internal"* %1 to i8*
  call void @llvm.memset.p0i8.i32(i8* %2, i8 0, i32 16, i1 false)
  %3 = bitcast %"kclassbody:A#internal"* %1 to %struct.ObjHeader*
  %typeInfoOrMeta  = getelementptr inbounds %struct.ObjHeader,
%struct.ObjHeader* %3, i32 0, i32 0
  store %struct.TypeInfo* @"kclass:A", %struct.TypeInfo** %typeInfoOrMeta_
  call void @"kfun:A#<init>(kotlin.Int){}"(%struct.ObjHeader* %3, i32 5)
  br label %when_exit

when_cond:                                        ; preds = %entry
  %4 = alloca %"kclassbody:A#internal"
  %5 = bitcast %"kclassbody:A#internal"* %4 to i8*
  call void @llvm.memset.p0i8.i32(i8* %5, i8 0, i32 16, i1 false)
  %6 = bitcast %"kclassbody:A#internal"* %4 to %struct.ObjHeader*
  %typeInfoOrMeta_1 = getelementptr inbounds %struct.ObjHeader,
%struct.ObjHeader* %6, i32 0, i32 0
  store %struct.TypeInfo* @"kclass:A", %struct.TypeInfo** %typeInfoOrMeta_1
  call void @"kfun:A#<init>(kotlin.Int){}"(%struct.ObjHeader* %6, i32 6)
  br label %when_exit

when_exit:
  %7 = phi %struct.ObjHeader* [ %3, %when_body ], [ %6, %when_cond ]
  call void @"kfun:A#print(){}"(%struct.ObjHeader* %7)
  br label %return_block

return_block:
  %8 = phi i32 [ 0, %when_exit ]
  ret i32 %8
}
```

Результаты тестов

Так как текущая реализация SSA IR поддерживает не все конструкции языка Котлин, вместо готовых наборов тестов пришлось создавать свой, в котором некоторые идиоматические конструкции языка были заменены более примитивными. Тесты были разбиты на 4 категории.

1. Примитивные тесты, в которых нет аллокаций. Данные тесты позволяют оценить то, насколько улучшается работа LLVM при изменении способа порождения входного языка. В отличие от старого пайплайна в новом LLVM IR сразу порождается в SSA-форме, что упрощает работу оптимизатора (табл. 2).

2. Тесты, в которых аллоцированные объекты не *убегают*. Эта категория позволяет получить верхнюю оценку прироста от escape-анализа, так как при его использовании нагрузка на сборщик мусора падает практически до нуля.

3. Тесты, в которых аллоцированные объекты *убегают*. Цель данной категории – оценить эффективность межпроцедурной компоненты escape-анализа.

4. Сложные тесты, близкие к реальному продуктовому коду. В них большое количество виртуальных вызовов, и активно используется стандартная библиотека.

Тестовый стенд: MacBook Pro 2015.

Процессор: Intel i7-4980HQ.

Оперативная память: 16 GB DDR3.
Операционная система: macOS 10.15.3.

Таблица 2

Сравнение производительности

Table 2

Performance comparison

Режим компиляции	Среднее время работы тестов, с			
	1	2	3	4
Без SSA IR	33.9	51.6	68.4	107.3
С SSA IR	30.3	5.2	46.7	90.5
Отношение производительности	1.12	9.92	1.46	1.18

Заключение

В данной работе рассмотрены архитектура компилятора Kotlin / Native и ее основные проблемы. Представлено новое внутреннее представление, которое обладает свойствами, необходимыми для написания продвинутых оптимизаций. На его основе был реализован ряд анализов и оптимизаций, включая построение графа вызова функций и escape-анализ. Была реализована трансляция в данное представление из существующего древовидного, а также трансляция в LLVM IR. Удалось сохранить совместимость с текущим процессом компиляции, что обеспечивает возможность постепенного перехода на новое представление. Эксперименты показали, что данный подход может значительно увеличить производительность порождаемого кода.

Список литературы / References

1. **Lattner C., Adve V.** LLVM: a compilation framework for lifelong program analysis & transformation. In: International Symposium on Code Generation and Optimization, 2004. CGO 2004, p. 75–86. DOI 10.1109/CGO.2004.1281665
2. **Bacon D. F., Cheng P., Rajan V. T.** A unified theory of garbage collection. In: Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, 2004. DOI 10.1145/1035292.1028982
3. **Appel A. W.** SSA is functional programming. *ACM SIGPLAN Notices*, April 1998, vol. 33, no. 4, p. 17–20. DOI 10.1145/278283.278285
4. **Braun M., Buchwald S., Hack S., Leißa R., Mallon C., Zwinkau A.** Simple and Efficient Construction of Static Single Assignment Form. *Compiler Construction. Lecture Notes in Computer Science*, 2013, vol. 7791, p. 102–122. DOI 10.1007/978-3-642-37051-9_6
5. **Cytron R., Ferrante J., Rosen B. K., Wegman M. N., Zadeck F. K.** Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, October 1991, vol. 13, no. 4, p. 451–490. DOI 10.1145/115372.115320
6. **Bacon D. F, Sweeney P. F.** Fast static analysis of C++ virtual function calls. In: Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. October 1996, p. 324–341. DOI 10.1145/236337.236371

7. **Choi J., Gupta M., Serrano M. J., Sreedhar V. C., Midkiff S. P.** Stack Allocation and Synchronization Optimizations for Java Using Escape Analysis. *ACM Transactions on Programming Languages and Systems*, November 2003, vol. 25, no. 6, p. 876–910. DOI 10.1145/945885.945892

Материал поступил в редколлегию
Received
03.06.2020

Сведения об авторе

Боголепов Сергей Сергеевич, магистрант факультета информационных технологий Новосибирского государственного университета (Новосибирск, Россия)
s.bogolepov@g.nsu.ru

Information about the Author

Sergey S. Bogolepov, Master's Student, Faculty of Information Technologies, Novosibirsk State University (Novosibirsk, Russian Federation)
s.bogolepov@g.nsu.ru

УДК 573.6
DOI 10.25205/1818-7900-2020-18-2-31-42

BioNet: моделирование масс-спектров пептидов

Р. Ю. Епифанов [1], **Д. А. Афонников** [1, 2]

[1] *Новосибирский государственный университет*
Новосибирск, Россия

[2] *Институт цитологии и генетики СО РАН*
Новосибирск, Россия

Аннотация

Определение белкового состава живой клетки (протеома) – одна из важнейших задач современной биологии. Универсальным инструментом для исследования протеома является масс-спектроскопия. Расшифровка масс-спектров является сложной задачей, так как не до конца известны механизмы диссоциации белков в экспериментальных установках, а также влияние совокупности внешних факторов на данный процесс. Для совершенствования существующих или разработки новых алгоритмов расшифровки масс-спектров требуется большое количество данных по аннотированным масс-спектрам пептидов с известной последовательностью. В статье описана разработка алгоритма *in silico* моделирования масс-спектра пептидов, решающего проблему учета влияния неканонического аминокислотного состава и посттрансляционных модификаций на процесс диссоциации. Для проверки работоспособности построенного алгоритма проведено сравнение его эффективности с аналогами. Показано, что точность предложенного метода выше, особенно для пептидов, подверженных посттрансляционным модификациям.

Ключевые слова

in silico масс-спектроскопия, нейросетевые методы, неканонические аминокислоты, посттрансляционные модификации

Благодарности

Работа поддержана грантами РФФИ № 17-00-00470 (К), 17-00-00462

Для цитирования

Епифанов Р. Ю., Афонников Д. А. BioNet: моделирование масс-спектров пептидов // Вестник НГУ. Серия: Информационные технологии. 2020. Т. 18, № 2. С. 31–42. DOI 10.25205/1818-7900-2020-18-2-31-42

BioNet: Peptide Mass-Spectrum Prediction

R. Yu. Epifanov [1], **D. A. Afonnikov** [1, 2]

[1] *Novosibirsk State University*
Novosibirsk, Russian Federation

[2] *Institute of Cytology and Genetics SB RAS*
Novosibirsk, Russian Federation

Abstact

The importance of the biological properties of proteins to cells cause actively exploring their amino acid composition (primary structure). The versatile tool of cell proteome exploring is mass-spectroscopy. The interpretation of mass-spectroscopy data is complex challenge because it remains uncertain peptide dissociation mechanisms and external factor influence to peptide fragmentation process. Moreover, a lot of mass-spectroscopy data is required to enhancement existing or development novel algorithms to interpret peptide mass-spectra. The article describes development of algorithm for *in silico* generation peptide mass-spectra covered the problem of influence noncanonical amino acid composition and posttranslational modifications to dissociation process. Developed algorithm was compared with analogues and evaluated over experimental data.

Keywords

in silico mass-spectroscopy, artificial neural networks, noncanonical amino acids, posttranslational modifications.

Acknowledgements

The research was supported by RFBR grants no. 17-00-00470(K), 17-00-00462

For citation

Epifanov R. Yu., Afonnikov D. A. BioNet: Peptide Mass-Spectrum Prediction. *Vestnik NSU. Series: Information Technologies*, 2020, vol. 18, no. 2, p. 31–42. (in Russ.) DOI 10.25205/1818-7900-2020-18-2-31-42

Введение

Белки являются важнейшими биологическими объектами. Разнообразие их функции в клетках организмов выше, чем у других биополимеров. В зависимости от выполняемых функций белки подразделяют на группы: ферменты, которые участвуют в специфическом катализе биологических реакций; структурные белки, которые определяют структуру тканей и форму тела животных; транспортные белки, участвующие в переносе веществ из клетки в клетку, и др.

Белки являются полимерами и состоят из аминокислот, связанных пептидными связями. Первичная структура белка может быть представлена в виде строки символов, каждый из которых представляет одну из 20 канонических аминокислот. Левый конец последовательности называется N-терминальным, правый – C-терминальным. Определение последовательности белка (его первичной структуры) является ключевым этапом в понимании механизмов его функционирования. Исследование первичной структуры белков возможно с помощью методов геномики. Секвенирование генома организма позволяет производить определение первичной структуры белка по последовательности кодирующих его генов [1]. Недостатком такого метода является то, что с помощью него невозможно определить наличие посттрансляционных модификаций в первичной структуре, которые оказывают важное влияние на механизмы функционирования белка.

Кроме того, отдельный исследовательский интерес вызывает группа пептидов нерибосомального пути биосинтеза благодаря их большому медицинскому значению. Среди них токсины, цитостатики, иммунодепрессоры [2]. В отличие от белков, кодируемых в геноме, синтез данной группы полипептидов происходит напрямую из аминокислот, что делает невозможным исследование первичной структуры методами геномики.

Это обусловливает исследование протеома клеток с помощью методов масс-спектроскопии. Тандемная масс-спектроскопия – это метод исследования, позволяющий установить первичную структуру пептидов [3]. В ее основе лежит (а) ферментативная фрагментация исследуемой смеси полипептидных цепей белков на короткие пептиды; (б) ионизация полученной смеси, при которой пептиды (их называют прекурсоры) получают заряд; (в) разделение этой пептидной смеси в ионной ловушке по отношению массы пептида m к его заряду z (m/z); (г) повторная диссоциация ионов-прекурсоров с образованием разнообразных структурно значимых ионных фрагментов, называемых вторичными ионами; (д) масс-анализ вторичных ионов. В результате такого разрушения исходной полипептидной цепи формируются серии спектров для каждого родительского иона. Каждый спектр – это распределение ионизированных фрагментов по величине отношения m/z и в графическом виде представляет собой набор узких пиков.

На следующем этапе осуществляется аннотация масс-спектров – сопоставление каждого из пиков масс-спектра пептида с известной структурной формулой. Аннотация позволяет определить пептидный состав белка. Информация о пептидном составе позволяет, в свою очередь, реконструировать последовательность исходного белка. Задача такой реконструкции является сложной, поскольку механизмы диссоциации пептидов до конца неизвестны, при аннотации установить структуру пептидов удается далеко не для всех спектральных пиков, на что отчасти влияет и совокупность внешних факторов, таких как величина заряда

прекурсора, величина энергии диссоциации и т. д. Дополнительные трудности создает посттрансляционная модификация белков при которой меняется химическая структура аминокислотных остатков уже после синтеза белка [4]. Посттрансляционная модификация существенно меняет значение *m*/*z* для аминокислот, кроме того, трудно заранее учесть все возможные варианты таких химических модификаций.

В настоящее время разработаны два типа алгоритмов реконструкции пептидной последовательности на основе масс-спектров. Первый основан на сопоставлении массы ионов в спектре с теоретическими масс-спектрами, рассчитанными на основании последовательностей всех возможных триптических пептидов для всех белков базы данных белковых последовательностей. Наличие уникальных теоретических пептидных фрагментов для какого-либо известного белка в спектре свидетельствует о его присутствии в смеси [3]. Ограничением таких алгоритмов является то, что в случае, если последовательность иона в базе отсутствует (что характерно для нерибосомальных пептидов), определение этого белка в смеси невозможно.

Алгоритмы сборки *de novo* работают на тех же принципах, что и алгоритмы сборки нуклеотидных последовательностей из фрагментов за счет их частичного перекрывания [1]. Они способны реконструировать последовательности пептидов, даже если те ранее были неизвестны. Однако процесс сборки требует большого времени счета (такая задача является NP-сложной [5]), и из-за того, что в спектрах всегда содержится шум (смещение линий из-за наличия изотопов, отсутствие некоторых линий, присутствие в спектре линий от загрязняющих веществ), получить решение весьма затруднительно. Также не ясна обобщающая способность алгоритмов *de novo* секвенирования для ранее неизвестных последовательностей.

Совершенствование существующих или разработка новых алгоритмов расшифровки масс-спектров требует много аннотированных масс-спектроскопических данных. Такие данные можно получить экспериментально, но этот способ обусловлен следующими недостатками: высокая трудоемкость, проведение экспериментов дорогостоящее, а аннотация масс-спектров часто проходит с невысокой достоверностью.

Другой способ получения таких данных – это компьютерное моделирование масс-спектров (масс-спектрометрия *in silico*). Данный подход не требует проведения трудоемких и дорогостоящих экспериментов, к тому же для масс-спектров оказываются известны исходные последовательности пептидов, что очень удобно для проверки работоспособности алгоритмов реконструкции.

В настоящее время разработан целый ряд методов, которые позволяют на основе аминокислотной последовательности пептида получить его спектр. Однако существующие алгоритмы моделирования масс-спектров не позволяют в полной мере проводить моделирование масс-спектров с учетом неканонического аминокислотного состава и посттрансляционных модификаций. Поэтому остается актуальной задача разработки алгоритмов моделирования масс-спектров пептидов.

В данной работе представлено построение алгоритма BioNet на основе методов искусственных нейронных сетей для моделирования масс-спектров пептидов с учетом неканонического аминокислотного состава и посттрансляционных модификаций. В работе проведено сравнение качества работы алгоритма с аналогами OpenMS-Simulator [6], MS2PIP [7] и pDeep [8].

Построение алгоритма BioNet

В настоящей работе мы предлагаем использовать для моделирования масс-спектров метод нейронных сетей. Нейронная сеть с точки зрения математики является вычислительным графом, в узлах которого содержатся операции или переменные, представляющие собой модели искусственных нейронов [9]. Графы искусственных нейронных сетей характеризуются наличием входного слоя, внутренних слоев и выходного слоя. Искусственные нейронные сети,

заданные таким образом, способны приблизить любую непрерывную функцию с любой требуемой точностью, в том числе и для задач обработки естественного языка – более общего случая обработки последовательностей.

В нашей работе, чтобы с помощью нейронной сети смоделировать по заданной аминокислотной последовательности масс-спектр, последовательность пептида необходимо представить в векторном виде, в котором ее можно будет поместить на входной слой вычислительного графа. Чтобы сохранять информацию о порядке аминокислот, удобно представлять последовательность пептида в виде последовательности признаков, которые уже кодируют отдельные аминокислоты. Использование признаков на основе векторов скрытого представления позволяет избежать недостатков вышеперечисленных подходов. В данной работе использовались векторы скрытого представления, полученные с помощью модели Mol2Vec [10].

Модель Mol2Vec в качестве предложения использует химическую структуру молекулы пептида, а для построения «слов» на основе этой структуры используется алгоритм Моргана [11], который для каждого атома производит идентификатор (рис. 1). Из этих идентификаторов составляется «предложение», которое характеризует молекулу. На вход модели Mol2vec подается химическая структура молекулы в формате SMILES [12].

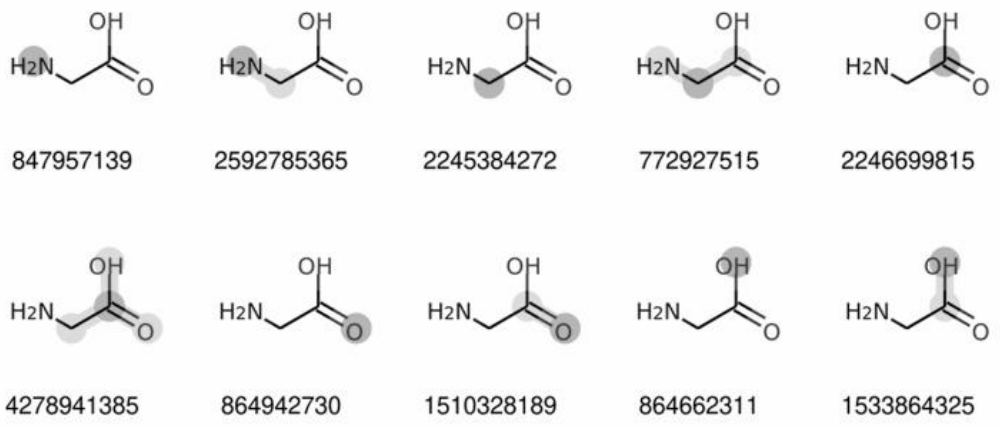

Рис. 1. Пример построения «слов» с помощью алгоритма Моргана для глицина. Идентификаторы расположены в том же порядке, что и атомы для канонического представления SMILES. Если атом имеет более чем один идентификатор, то они расположены в порядке возрастания радиуса (расстояние между атомами в графе молекулы). Адаптировано из [12]

Fig. 1. The example of construction “words” with Morgan algorithm for glycine. Identifiers is placed the same order as atoms in SMILES canonical representation of molecule. If atom has more than one identifier, then they are placed in ascending order. Adapted from [12]

В данной работе использована модель Mol2Vec типа Skipgram с размером контекста 10, радиусом Моргана 1 и размерностью векторов скрытого пространства 300. Модель доступна по адресу github.com/samoturk/mol2vec.

После определения способа векторизации последовательностей пептидов дальнейший поиск архитектуры необходимо было проводить в терминах задачи моделирования последовательности по последовательности. В контексте решаемой задачи требуется по аминокислотной последовательности пептида моделировать интенсивности ионов основных серий, получаемых в ходе масс-спектроскопического эксперимента. В таких задачах моделирования хорошо себя зарекомендовали архитектуры нейронных сетей на рекуррентных моделях [13; 14].

Поскольку на величину энергии разрыва связи в пептиде влияют аминокислоты как со стороны N-конца, так и со стороны C-конца [8], принято решение использовать двунаправленный вариант LSTM слоев, чтобы учесть данную особенность. Кроме того, на величину энергии также оказывают влияние концевые аминокислоты, поэтому логично передавать на вход нейронной сети не только последовательность пептида, но и бинарную маску, в которой будут помечены концевые аминокислоты последовательности.

В [15] показано, что обучаемая линейная комбинация выходов слоев дает лучшее представление для решения задачи. Такая же техника использована в данной работе. Итоговая архитектура нейронной сети для моделирования масс-спектров представлена на рис. 2.

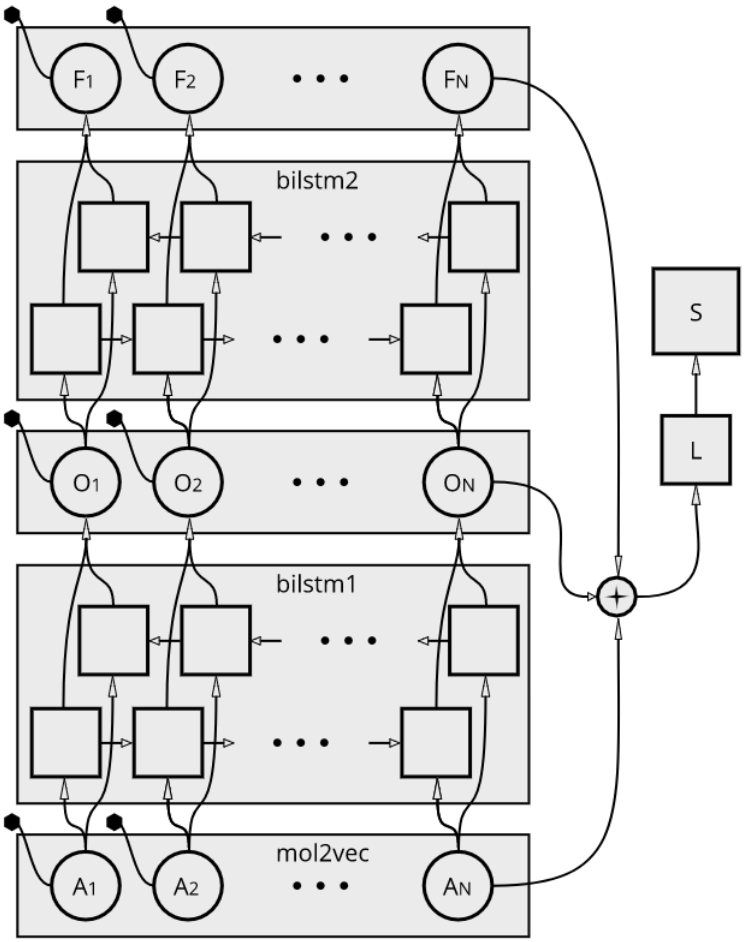

Рис. 2. Архитектура нейронной сети для моделирования масс-спектров пептидов: Ai – аминокислота; Oi – выход первого двунаправленного слоя; Fi – выход второго двунаправленного слоя; mol2vec – слой сети для получения векторов скрытого представления для аминокислот; bilstm – двунаправленный рекуррентный слой; + – обучаемая линейная комбинация, • – действие обучаемой линейной комбинации; L – группа линейный слоев; S – смоделированный спектр

Fig. 2. Neural network architecture to predict peptide mass-spectra. Ai – amino acid, Oi – first bidirectional layer output, Fi – second bidirectional layer output, mol2vec – amino acid embedding layer, bilstm – bidirectional recurrent layer, + – learnable linear combination, • – action learnable linear combination, L – stacked linear layer, S – predicted mass-spectrum

Описание метрики качества и функции потерь для обучения алгоритма

В задаче попарного сравнения масс-спектров принято использовать следующие метрики: косинусное расстояние [8; 16; 17], коэффициенты корреляции Пирсона [7; 8] или Спирмена [8]. В данной работе используется коэффициент корреляции Пирсона (ККП) для попарного сравнения масс-спектров. ККП характеризует существование линейной зависимости между двумя величинами.

Пусть дана аминокислотная последовательность, известен экспериментальный масс-спектр y и смоделирован теоретический масс-спектр p.

ККП рассчитывается по формуле

$$pearsim\ \ y, p\ \ = \frac{\sum_j \left(y_j - \underline{y}\right)\left(p_j - \underline{p}\right)}{\left[\sum_j^2 \left(y_j - \underline{y}\right)\right]^{0.5}\left[\sum_j^2 \left(p_j - \underline{p}\right)\right]^{0.5}},$$

где $\underline{x} = \frac{\sum_{i=1}^{N} x_i}{N}$.

Оценки качества моделирования масс-спектров для набора D пар масс-спектр-пептид происходит следующим образом:

$$pD = pearson(D),$$
$$pD = sort(pD),$$
$$mpD = median(pD),$$
$$rpD = \frac{1}{|pD|}\sum_{i\ pD_i > 0.75},$$

где mpD – медианное значение (мККП), а rpD – доля больше 0,75 (дККП) для набора ККП. Величины мККП и дККП характеризуют качество моделирования масс-спектров: чем они больше, тем выше точность моделирования масс-спектров.

Для обучения сети использована функция потерь следующего вида:

$$loss = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - p_i\right)^2 + \frac{1}{N}\sum_{i=1}^{N}\frac{\sum_j \left(y_{ij} - \underline{y_i}\right)\left(p_{ij} - \underline{p_i}\right)}{\left[\sum_j^2 \left(y_{ij} - \underline{y_i}\right)\right]^{0.5}\left[\sum_j^2 \left(p_{ij} - \underline{p_i}\right)\right]^{0.5}} + \sum_k \frac{c}{w_k^2},$$

где y_i – экспериментальный масс-спектр, p_i – теоретический масс-спектр, w_k – веса нейронной сети, C – константа регуляризации, N – размер обучающей выборки. Первый член функции потерь оптимизирует абсолютные значения смоделированного спектра, второй член оптимизирует метрику качества, последний член необходим для регуляризации весов модели.

Выбор данных для обучения алгоритма

В качестве источника данных для тестирования алгоритма BioNet использовались масс-спектры из базы данных PRIDE [18], проекты PXD004732 и PXD000138. Проект PXD004732 содержит около четырех миллионов аннотированных тандемных масс-спектров пептидов с каноническим аминокислотным составом [19]. Этот набор данных получен в рамках проек-

та Proteome Tools. Масс-спектры Proteome Tools получены в результате расщепления пептидов человека трипсином, дальнейшей диссоциации на разных режимах с последующим использованием различных вариантов детекторов ионов. Полученные масс-спектры аннотировались с помощью программы Andromeda [20], которая является поисковой системой, использующей информацию о достоверно определенных масс-спектрах пептидов для аннотации неизвестных масс-спектров.

Проект PXD000138 содержит около двухсот тысяч масс-спектров, полученных методом диссоциации HCD, для пептидов с каноническим аминокислотным составом и их аналогов с модификациями типа оксидирование (M), фосфорилирование (ST) и фосфорилирование (Y) [21]. Масс-спектры в рамках данного проекта также аннотировались с помощью программы Andromeda [20].

Обработка данных

Для загруженных данных были построены распределения пептидов по длине и заряду (рис. 3, 4), для того чтобы оценить эти две основные характеристики пептидных данных.

На основе построенных распределений было решено ограничить размер входной последовательности двадцатью аминокислотами и исключить из рассмотрения масс-спектры зарядом прекурсора один, пять и более из-за их малой представленности в данных. Ранее в работе Zolg и др. [22] было предложено использовать только результаты аннотации с коэффициентом уверенности (Andromeda score) больше 100. Мы использовали этот порог для фильтрации наших данных. Кроме того, масс-спектр исключался из рассмотрения, если количество аннотированных пиков было меньше, чем количество аминокислот в пептиде.

После фильтрации данных по количеству аминокислот в пептиде, заряду прекурсора, качеству аннотации и количеству пиков из проекта PXD004732 было получено 2 325 000 масс-спектров, из проекта PXD000138 – 91 230 масс-спектров, среди них 36 740 масс-спектров с модифицированными аминокислотными последовательностями.

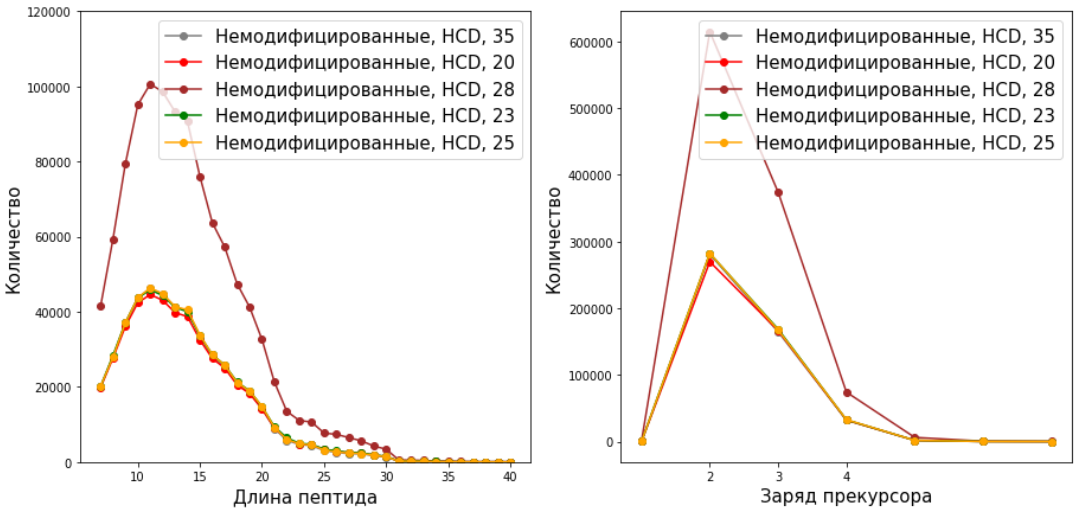

Рис. 3. График распределения пептидов по длине (слева) и заряду прекурсора (справа) для данных проекта PXD004732

Fig. 3. Peptide distribution plots by length (left) and precursor charge (right) over PXD004732 data

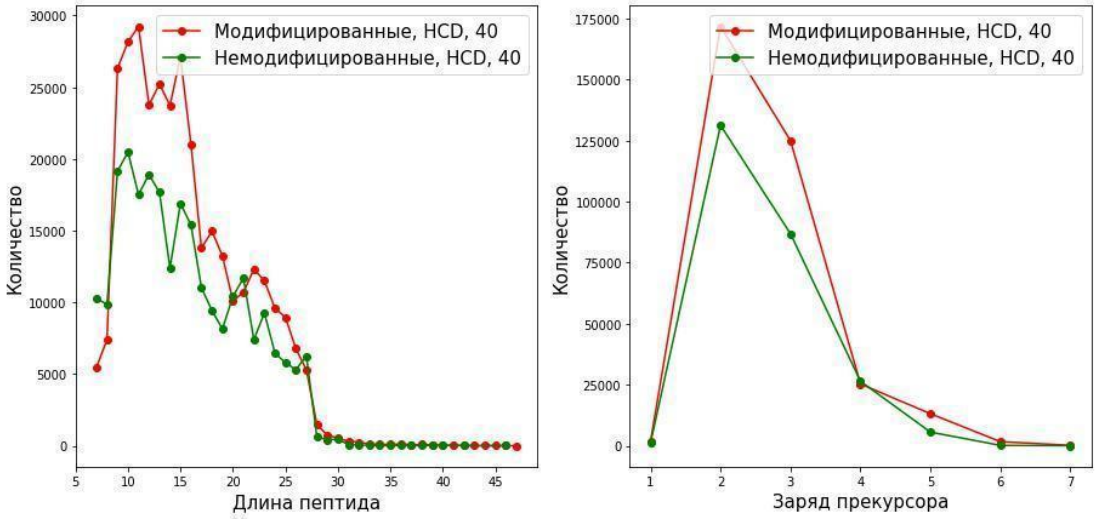

Рис. 4. График распределения пептидов по длине (слева) и заряду прекурсора (справа) для данных проекта PXD000138

Fig. 4. Peptide distribution plots by length (left) and precursor charge (right) over PXD000138 data

Для обучения было отобрано 80 % данных с каноническим аминокислотным составом, остальные 20 % были отложены для тестирования, также 100 % данных с неканоническим аминокислотным составом были использованы для тестирования.

Точность моделирования масс-спектров

Результатом работы нашего алгоритма является смоделированный масс-спектр пептида, поэтому эффективность работы алгоритма формулируется в терминах мКПП и дКПП, описанных ранее. Чем выше эти значения, тем точнее работает алгоритм.

мККП и дККП, полученные в результате тестирования разработанного метода, приведены в табл. 1, 2. Данные представлены в сравнении с алгоритмами OpenMS-Simulator, MS2PIP и pDeep. Также для данных была получена теоретическая оценка максимально достижимого качества [8], расчет проводился для последовательностей, представленных несколькими масс-спектрами, моделированным масс-спектром считался усредненный масс-спектр. Пример смоделированных масс-спектров с ККП 0.992 и 0.718 приведен на рис. 5.

Таблица 1

Результаты тестирования для PDX000138, PDX004732, канонические аминокислоты

Table 1

The results of evaluation over PDX000138, PDX004732 data with canonical amino acid composition

	мККП	дККП
OpenMS-Simulator	0.710	44.05
MS2PIP	0.845	71.80
pDeep	0.246	4.36
Предложенный алгоритм	0.909	76.78
Верхняя оценка	0.986	99.33

Таблица 2

Результаты тестирования для PDX000138, посттрансляционные модификации (фосфорилирование по S, T, Y, оксидирование по M) *

Table 2

The results of evaluation over PDX000138 data with posttranslational modifications (phospho S, T, Y, oxidation M) **

	мККП	дККП
OpenMS-Simulator	0.710	44.05
MS2PIP	0.845	71.80
pDeep	0.246	4.36
Предложенный алгоритм	0.909	76.78
Верхняя оценка	0.986	99.33

* Приведены данные моделирования без учета модификаций.
** Data are predicted disregarding posttranslational modifications.

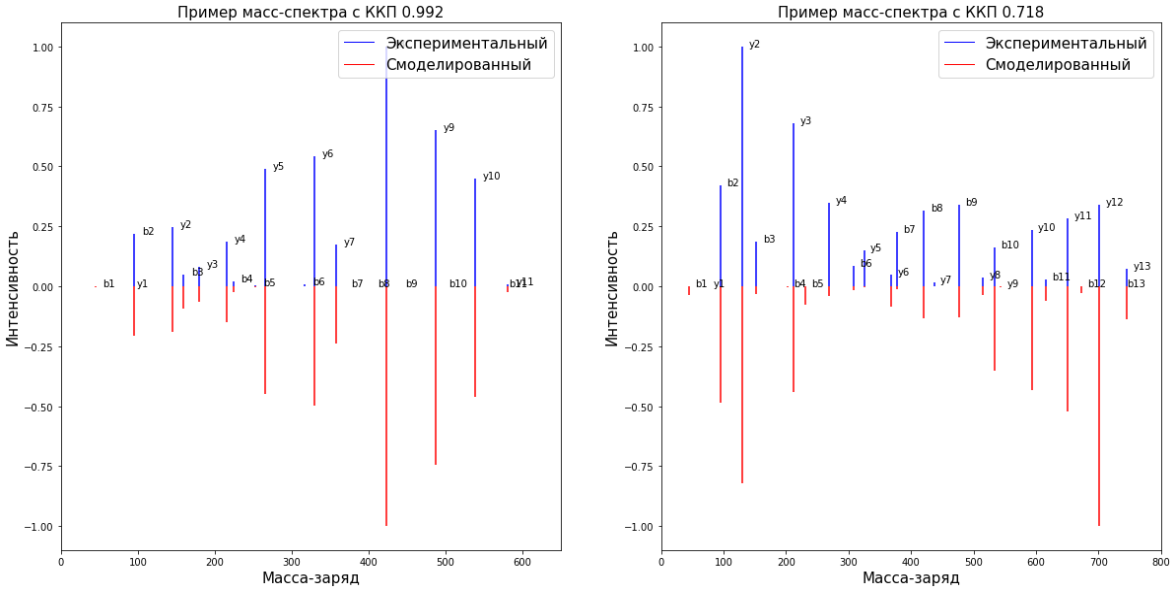

Рис. 5. Пример моделирования масс-спектра предложенным алгоритмом. Сверху расположен экспериментальный масс-спектр, снизу теоретический. Спектры представлены для последовательностей STEEGEVAALR (слева) и STDVGRHSLLYLK (справа)

Fig. 5. The example of predicting mass-spectrum by developed algorithm. From above it is placed experimental, from below is placed theoretical mass-spectrum. Spectra are shown for STEEGEVAALR (left) and STDVGRHSLLYLK (right) sequences

Заключение

В рамках выполнения работы были получены следующие результаты.

1. Реализован алгоритм BioNet для моделирования масс-спектров пептидов на основе нейронной сети глубокого обучения.

2. Проведено сравнение разработанного метода и трех аналогичных алгоритмов для задачи моделирования масс-спектров пептидов с каноническим аминокислотным составом. Разработанный алгоритм показал лучшее качество моделирования.

3. Проведено сравнение разработанного метода для моделирования масс-спектров с неканоническим аминокислотным составом. Разработанный алгоритм показал наиболее высокое качество моделирования спектров.

Список литературы

1. **Задесенец К. С., Ершов Н. И., Рубцов Н. Б.** Полногеномное секвенирование геномов эукариот: от секвенирования фрагментов ДНК к сборке генома // Генетика. 2017. Т. 53, № 6. С. 641–650.
2. **Орлова Т. И., Булгакова В. Г., А. Н. Полин.** Биологически активные нерибосомальные пептиды. III. Нерибосомальные антибиотики полипептиды // Антибиотики и химиотерапия. 2012. № 7. С. 43–54.
3. **Краснов Н. В., Лютвинский Я. И., Подольская Е. П.** Масс-спектрометрия с мягкими методами ионизации в протеомном анализе (Обзор) // Научное приборостроение. 2010. Т. 20, № 4. С. 5–20.
4. **Кнорре Д. Г., Кудряшова Н. В., Годовикова Т. С.** Химические и функциональные аспекты посттрансляционной модификации белков // Acta Naturae. 2009. Т. 1, № 3.
5. **Fomin E. A.** Simple Approach to the Reconstruction of a Set of Points from the Multiset of n 2 Pairwise Distances in n 2 Steps for the Sequencing Problem: I. Theory. *Journal of Computational Biology*, 2016, vol. 23, no. 9, p. 769–775.
6. **Wang Y. et al.** OpenMS-Simulator: an open-source software for theoretical tandem mass spectrum prediction. *BMC bioinformatics*, 2015, vol. 16, no. 1, p. 110.
7. **Degroeve S., Martens L.** MS2PIP: a tool for MS/MS peak intensity prediction. *Bioinformatics*, 2013, vol. 29, no. 24, p. 3199–3203.
8. **Zhou X. X. et al.** pDeep: predicting MS/MS spectra of peptides with deep learning. *Analytical chemistry*, 2017, vol. 89, no. 23, p. 12690–12697.
9. **Созыкин А. В.** Обзор методов обучения глубоких нейронных сетей // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2017. Т. 6, № 3. С. 28–59. DOI 10.14529/cmse170303
10. **Jaeger S., Fulle S., Turk S.** Mol2vec: unsupervised machine learning approach with chemical intuition. *Journal of chemical information and modeling*, 2018, vol. 58, no. 1, p. 27–35.
11. **Rogers D., Hahn M.** Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 2010, vol. 50, no. 5, p. 742–754.
12. **Richard L. Rowley, R. Jeremy Rowley, John L. Oscarson, W. Vincent Wilding.** Development of an Automated SMILES Pattern Matching Program to Facilitate the Prediction of Thermo physical Properties by Group Contribution Methods. Department of Chemical Engineering. Brigham Young University. Provo, Utah, 2001, p. 1110–1113.
13. **Sutskever I., Vinyals O., Le Q. V.** Sequence to Sequence Learning with Neural Networks. In: arXiv preprint arXiv:1409.3215. 2014.
14. **Graves A.** Generating sequences with recurrent neural networks. In: arXiv preprint arXiv:1308.0850. 2013.
15. **Peters M. E. et al.** Deep contextualized word representations. In: arXiv preprint arXiv:1802.05365. 2018.
16. **Dong N. et al.** Prediction of peptide fragment ion mass spectra by data mining techniques. *Analytical chemistry*, 2014, vol. 86, no. 15, p. 7446–7454.
17. **Lin Y. M., Chen C. T., Chang J. M.** MS2CNN: predicting MS/MS spectrum based on protein sequence using deep convolutional neural networks. *BMC genomics*, 2019, vol. 20, no. 9, p. 1–10.
18. **Perez-Riverol Y. et al.** The PRIDE database and related tools and resources in 2019: improving support for quantification data. *Nucleic acids research*, 2019, vol. 47, no. D1, p. D442–D450.

19. **Gessulat S. et al.** Prosit: proteome-wide prediction of peptide tandem mass spectra by deep learning. *Nature methods*, 2019, vol. 16, no. 6, p. 509.
20. **Cox J. et al.** Andromeda: a peptide search engine integrated into the MaxQuant environment. **Journal of proteome research**, 2011, vol. 10, no. 4, p. 1794–1805.
21. **Marx H. et al.** A large synthetic peptide and phosphopeptide reference library for mass spectrometry based proteomics. *Nature biotechnology*, 2013, vol. 31, no. 6, p. 557–564.
22. **Zolg D. P. et al.** Building Proteome Tools based on a complete synthetic human proteome. *Nature methods*, 2017, vol. 14, no. 3, p. 259–262.

References

1. **Zadesenets K. S., Yershov N. I., Rubtsov N. B.** Genome-Wide sequencing of genomes eukaryote: from sequencing of DNA fragments for Assembly of the genome. *Genetics*, 2017, vol. 53, no. 6, p. 641–650. (in Russ.)
2. **Orlova T. I., Bulgakova V. G., Polin A. N.** Biologically active non-ribosomal peptides. III. Non-ribosomal antibiotics polypeptides. *Antibiotics and Chemotherapy*, 2012, no. 7, p. 43–54. (in Russ.)
3. **Krasnov N. V., Lyutvinsky Ya. I., Podolskaya E. P.** Soft mass spectrometry methods of ionization in proteomic analysis (Review). Scientific Instrumentation, 2010, vol. 20, no. 4, p. 5–20. (in Russ.)
4. **Knorre D. G., Kudryashova N. V., Godovikova T. S.** Chemical and functional aspects of posttranslational modification of proteins. *Acta Naturae*, 2009, vol. 1, no. 3. (in Russ.)
5. **Fomin E. A.** Simple Approach to the Reconstruction of a Set of Points from the Multiset of n 2 Pairwise Distances in n 2 Steps for the Sequencing Problem: I. Theory. *Journal of Computational Biology*, 2016, vol. 23, no. 9, p. 769–775.
6. **Wang Y. et al.** OpenMS-Simulator: an open-source software for theoretical tandem mass spectrum prediction. *BMC bioinformatics*, 2015, vol. 16, no. 1, p. 110.
7. **Degroeve S., Martens L.** MS2PIP: a tool for MS/MS peak intensity prediction. *Bioinformatics*, 2013, vol. 29, no. 24, p. 3199–3203.
8. **Zhou X. X. et al.** pDeep: predicting MS/MS spectra of peptides with deep learning. *Analytical chemistry*, 2017, vol. 89, no. 23, p. 12690–12697.
9. **Sozykin A. V.** Review of training methods for deep neural networks. *Bulletin of SUSU. Series: Computational mathematics and computer science*, 2017, vol. 6, no. 3, p. 28–59. (in Russ.) DOI 10.14529/cmse170303
10. **Jaeger S., Fulle S., Turk S.** Mol2vec: unsupervised machine learning approach with chemical intuition. *Journal of chemical information and modeling*, 2018, vol. 58, no. 1, p. 27–35.
11. **Rogers D., Hahn M.** Extended-connectivity fingerprints. *Journal of chemical information and modeling*, 2010, vol. 50, no. 5, p. 742–754.
12. **Richard L. Rowley, R. Jeremy Rowley, John L. Oscarson, W. Vincent Wilding.** Development of an Automated SMILES Pattern Matching Program to Facilitate the Prediction of Thermo physical Properties by Group Contribution Methods. Department of Chemical Engineering. Brigham Young University. Provo, Utah, 2001, p. 1110–1113.
13. **Sutskever I., Vinyals O., Le Q. V.** Sequence to Sequence Learning with Neural Networks. In: arXiv preprint arXiv:1409.3215. 2014.
14. **Graves A.** Generating sequences with recurrent neural networks. In: arXiv preprint arXiv:1308.0850. 2013.
15. **Peters M. E. et al.** Deep contextualized word representations. In: arXiv preprint arXiv:1802.05365. 2018.
16. **Dong N. et al.** Prediction of peptide fragment ion mass spectra by data mining techniques. *Analytical chemistry*, 2014, vol. 86, no. 15, p. 7446–7454.

17. **Lin Y. M., Chen C. T., Chang J. M.** MS2CNN: predicting MS/MS spectrum based on protein sequence using deep convolutional neural networks. *BMC genomics*, 2019, vol. 20, no. 9, p. 1–10.
18. **Perez-Riverol Y. et al.** The PRIDE database and related tools and resources in 2019: improving support for quantification data. *Nucleic acids research*, 2019, vol. 47, no. D1, p. D442–D450.
19. **Gessulat S. et al.** Prosit: proteome-wide prediction of peptide tandem mass spectra by deep learning. *Nature methods*, 2019, vol. 16, no. 6, p. 509.
20. **Cox J. et al.** Andromeda: a peptide search engine integrated into the MaxQuant environment. **Journal of proteome research**, 2011, vol. 10, no. 4, p. 1794–1805.
21. **Marx H. et al.** A large synthetic peptide and phosphopeptide reference library for mass spectrometry based proteomics. *Nature biotechnology*, 2013, vol. 31, no. 6, p. 557–564.
22. **Zolg D. P. et al.** Building Proteome Tools based on a complete synthetic human proteome. *Nature methods*, 2017, vol. 14, no. 3, p. 259–262.

Материал поступил в редколлегию
Received
05.06.2020

Сведения об авторах

Епифанов Ростислав Юрьевич, магистрант 2-го курса факультета информационных технологий Новосибирского национального исследовательского государственного университета (Новосибирск, Россия)
rostepifanov@gmail.com

Афонников Дмитрий Аркадьевич, кандидат биологических наук, ведущий научный сотрудник Института цитологии и генетики СО РАН (Новосибирск, Россия)
ada@bionet.nsc.ru

Information about the Authors

Rostislav Yu. Epifanov, Master Student 2yr, Faculty of Information Technologies, Novosibirsk State University (Novosibirsk, Russian Federation)
rostepifanov@gmail.com

Dmitry A. Afonnikov, PhD of Biology, Leading Research Scientist, ICG SB RAS (Novosibirsk, Russian Federation)
ada@bionet.nsc.ru

УДК 004.438
DOI 10.25205/1818-7900-2020-18-2-43-53

Инструменты для выполнения и эмуляции квантовых вычислений

П. Е. Баскаков, Ю. Ю. Хабовец, И. А. Пилипенко
В. О. Кравченко, Л. В. Черкесова

Донской государственный технический университет
Ростов-на-Дону, Россия

Аннотация

В настоящее время квантовые технологии находятся на передовой развития научной мысли. Крупные корпорации создают собственные квантовые суперкомпьютеры, разрабатываются квантовые аналоги классических алгоритмов, ведутся исследования в области квантовой криптографии. Но так как квантовые компьютеры еще не получили широкого распространения, актуальным становится вопрос: как обычным пользователям, ученым и исследователям не отставать от развития науки? Одним из возможных решений является использование различного рода инструментов для эмуляции квантовых вычислений на локальном неквантовом компьютере. Кроме того, существует также возможность получить в распоряжение несколько кубит квантового суперкомпьютера IBM. Как правило, такие инструменты реализуются в виде библиотек для различных языков программирования. Ввиду того что работа с реальными квантовыми компьютерами доступна лишь узкому кругу исследователей, эмуляторы просто необходимы для проверки гипотез или алгоритмов. В данной статье рассмотрены наиболее популярные квантовые эмуляторы, использующиеся для квантовых вычислений и позволяющие эмулировать процесс работы квантового компьютера. Были изучены квантовые эмуляторы, выявлены и описаны их индивидуальные особенности, составлены рекомендации для более удобного начала работы с ними, а также описаны их достоинства и недостатки. Произведен обзор нескольких библиотек для языков JavaScript, Python, C/C++, а также рассмотрены средство с веб-интерфейсом (Quantum Programming Studio) и набор инструментов от компании Microsoft (Microsoft Quantum Development Kit), основным языком которого служит Q#.

Ключевые слова

квантовые вычисления, библиотеки, эмуляция, кубит, языки программирования

Для цитирования

Баскаков П. Е., Хабовец Ю. Ю., Пилипенко И. А., Кравченко В. О., Черкесова Л. В. Инструменты для выполнения и эмуляции квантовых вычислений // Вестник НГУ. Серия: Информационные технологии. 2020. Т. 18, № 2. С. 43–53. DOI 10.25205/1818-7900-2020-18-2-43-53

Tools for Performing and Emulating Quantum Computing

P. E. Baskakov, Yu. Yu. Khabovets, I. A. Pilipenko
V. O. Kravchenko, L. V. Cherkesova

Don State Technical University
Rostov on Don, Russian Federation

Abstract

Currently, quantum technologies are at the forefront of scientific thought. Large corporations are creating their own quantum supercomputers, developing quantum analogues of classical algorithms, and research is being conducted in the field of quantum cryptography. But since quantum computers have not yet become widespread, the question be-

comes relevant: how can ordinary users, scientists and researchers keep up with the development of science? One possible solution is to use various kinds of tools to emulate quantum computing on a local non-quantum computer. In addition, there is also the opportunity to have several qubits of IBM's quantum supercomputer available. As a rule, such tools are implemented in the form of libraries for various programming languages. Due to the fact that working with real quantum computers is available only to a narrow circle of researchers, emulators are simply necessary to test hypotheses or algorithms. This article examined the most popular quantum emulators used for quantum computing and allowing emulating the process of a quantum computer. Work was carried out to study quantum emulators, to identify and describe their individual characteristics, to make recommendations for a more convenient start to work with them, as well as to describe their advantages and disadvantages. A review of several libraries for the JavaScript, Python, C / C ++ languages was made, as well as a tool with a web interface (Quantum Programming Studio) and a set of tools from Microsoft (Microsoft Quantum Development Kit), the main language of which is Q #, is examined. At the end of the article, a conclusion is made regarding the considered tools.

Keywords

quantum computations, libraries, emulation, qubit, programming languages

For citation

Baskakov P. E., Khabovets Yu. Yu., Pilipenko I. A., Kravchenko V. O., Cherkesova L. V. Tools for Performing and Emulating Quantum Computing. *Vestnik NSU. Series: Information Technologies*, 2020, vol. 18, no. 2, p. 43–53. (in Russ.) DOI 10.25205/1818-7900-2020-18-2-43-53

Введение

Идея применения квантовых механизмов для создания устройств, названных впоследствии квантовыми компьютерами, была подана советским ученым Юрием Маниным в работе «Вычислимое и невычислимое» [1], позднее в работе «Simulating physics with computers» Ричард Фейнман продолжил эти рассуждения [2]. Квантовый компьютер является единственной на сегодняшний день моделью, способной предложить экспоненциальный прирост скорости вычислений по сравнению с обычными современными компьютерами, пусть это и касается лишь ограниченного круга задач [3].

В статье «Quantum supremacy using a programmable superconducting processor» вводится такое понятие, как «квантовое превосходство». Это потенциальная способность квантовых вычислительных устройств решать проблемы, которые классические компьютеры практически не могут, или им требуется для этого слишком много времени [4]. В той же статье был дан критерий наступления квантового превосходства: по мнению авторов, это порог вычислительной мощности в 50 кубитов. Таким образом, квантовое превосходство было достигнуто, а недавно созданная 72-кубитная квантовая машина Google Bristlecone уже превосходит классический суперкомпьютер по четко определенной вычислительной задаче. Но, к сожалению, таких компьютеров в мире всего несколько, поэтому большинству исследователей еще долго придется довольствоваться эмуляторами квантовых вычислений.

Целью данной работы является обзор современных инструментов для работы с квантовыми вычислениями и компьютерами. В данной статье можно ознакомиться с существующим инструментарием разработчика квантовых приложений, прикоснуться к реальным квантовым вычислениям и выбрать для себя наиболее подходящий инструмент для работы с квантовым эмулятором или компьютером.

1. **Quantum circuit** – это библиотека с открытым исходным кодом, предназначенная для эмуляции квантовых вычислений в программах на языке программирования JavaScript. Благодаря этому можно его использовать прямо в браузере при создании html-страницы либо на сервере node.js. Также можно применять quantum cirquit внутри Jupyter Notebook, однако в этом случае понадобится установка дополнительного плагина. Созданные квантовые схемы доступны для импорта / экспорта в форматы других распространенных эмуляторов (Qiskit, Cirq, QuEST и др.) и для сохранения в виде векторных рисунков.

По заверениям разработчиков библиотеки, quantum cirquit способен обеспечивать эмуляцию более 20 кубит без существенного влияния на производительность системы [1]. Для этого используется ряд «ухищрений», призванных оптимизировать потребление оперативной памяти. В частности, алгоритм симуляции не хранит полный вектор состояний в памяти в виде массива размера 2^n, где n – количество кубит в схеме, а использует специальную структуру данных, в которой сохранены только ненулевые значения. Элементы матрицы преобразований рассчитываются, умножаются на вектор состояний и сохраняются «на лету», таким образом позволяя достичь единовременного хранения максимум 2 векторов состояний. Алгоритм допускает распараллеливание, то есть проведение расчетов возможно в том числе с использованием графического ускорителя (Graphics Processing Unit, GPU), однако данная особенность в настоящее время не реализована.

Рассмотрим теперь пример работы с библиотекой на основе следующего листинга кода на языке JavaScript:

```
var circuit = new QuantumCircuit(2);
circuit.addGate("h", 0, 1);
circuit.addMeasure(1, "c", 0);
circuit.run();
console.log(circuit.getCregValue("c")).
```

В данном примере создается схема с двумя кубитами, ко второму из которых применяется преобразование Адамара с последующим измерением кубита и сохранением результата в 0-й бит классического не квантового регистра *c*, после чего схема запускается на выполнение и на экран выводится значение регистра. Следует отметить, что нужные регистры будут добавлены на схему автоматически, если ранее этого не было сделано явным образом. Полный список реализованных преобразований доступен на странице документации библиотеки [2].

Для определения сравнительных признаков рациональным будет выделить достоинства и недостатки пакета quantum–circuit.

К достоинствам можно отнести:

- открытость исходного кода;
- хорошую документированность;
- возможность интеграции с форматами других библиотек и оптимизированное потребление памяти.

Главным недостатком является несбалансированное использование ресурсов вычислительной системы [3]. Так, реализация проведения вычислений на GPU позволила бы снять часть нагрузки с процессора и сократить время, требуемое на обработку.

2. **QuEST.** Вышеописанный недостаток устранен в другой популярной библиотеке квантовых вычислений – QuEST (Quantum Exact Simulation Toolkit), разрабатываемой специалистами исследовательской группы Оксфордского университета QTechTheory.

Данная библиотека предназначена для использования совместно с языками программирования C / C++. Благодаря этому достигается возможность использования ее на любой целевой платформе – от ноутбуков до суперкомпьютеров, требуется лишь наличие соответствующего компилятора.

Библиотека направлена на точную и эффективную симуляцию глубоких квантовых схем с большим числом кубитов. Отличительной особенностью является возможность построения распределенной сети, что позволяет объединить вычислительные мощности нескольких ком-

[1] Quantum Circuit Simulator. 2020. URL: https://www.npmjs.com/package/quantum-circuit (дата обращения 15.03.2020).

[2] Там же.

[3] List of QC simulators / Quantiki – Portal and Wiki, 2020. URL: https://quantiki.org/wiki/list-qc-simulators (дата обращения 12.04.2020).

пьютеров-узлов для достижения наибольшей производительности. Так, с использованием 2 048 компьютеров ARCUS и ARCHER удалось смоделировать 38 кубит [5].

Алгоритмы симуляции являются многопоточными, задействуются не только мощности центрального процессора, но и GPU, для чего требуется наличие библиотек Nvidia CUDA (Compute Unified Device Architecture). Сообщается, что с использованием видеоадаптера с объемом памяти 2 Гб возможно моделирование 26 кубитов, на ноутбуке с 16 Гб оперативной памяти – 29 кубитов [4].

Работа с библиотекой должна начинаться с вызова функции *createQuESTEnv*, которая вернет объект класса *QuESTEnv* – окружение, инкапсулирующее конфигурацию многопоточности, распределенности и GPU-ускорения.

Рассмотрим следующий код на языке программирования C++:

```
QuESTEnv env = createQuESTEnv();
Qureg qubits = createQureg(3, env);
initZeroState(qubits);
hadamard(qubits, 0);
controlledNot(qubits, 0, 1);
rotateY(qubits, 2, .1);
destroyQureg(qubits, env);
destroyQuESTEnv(env).
```

Квантовые регистры создаются вызовом функции *createQreg*. Затем регистры могут быть инициализированы нулевым состоянием (*initZeroState*) либо положительным (*initPlusState*).

Далее к регистрам можно применять преобразования (которые тут называются *гейтами*). Список реализованных гейтов доступен по ссылке [5].

При завершении работы рекомендуется освободить память, выделенную под регистры и окружение (функции *destroyQureg* и *destroyQuESTEnv* соответственно).

На основании вышесказанного можно сделать вывод, что главное преимущество QuEST заключается в комбинировании производительности центрального процессора (Central Processing Unit, CPU) и GPU, а также возможности проведения распределенных вычислений. На данный момент это единственный симулятор с открытым исходным кодом, обладающий подобным функционалом [5]. Также следует отметить подробную документированность и наличие примеров схем в официальном репозитории.

3. **Qiskit** – фреймворк для квантовых вычислений с открытым исходным кодом, разрабатываемый исследовательской группой IBM Research, а также сообществом энтузиастов с целью создания ПО для облачных квантовых вычислений.

Основная версия Quiskit использует Python в качестве языка программирования, однако доступны также версии для языков Swift и JavaScript [6]. Quiskit предоставляет возможность разработки квантового ПО как на высоком уровне абстракции (для пользователей без опыта квантового программирования), так и на низком уровне, близком к машинному коду OpenQASM. Такая возможность обеспечивается благодаря использованию 4-х компонентов: Terra, Aqua, Aer, Ignis.

- Qiskit Terra является своего рода ядром фреймворка [7]. Данный модуль предоставляет инструменты для создания квантовых схем на машинном или близком к нему уровне, а также квантовых гейтов. Кроме этого, Qiskit Terra содержит инструменты для оптимизации кванто-

[4] Quantum Exact Simulation Toolkit, 2020. URL: https://quest.qtechtheory.org/about/ (дата обращения 02.04.2020).

[5] QuEST Coding / Quantum Exact Simulation Toolkit, 2020. URL: https://quest.qtechtheory.org/docs/#coding (дата обращения 02.04.2020).

[6] Qiskit / Wikipedia – The free Encyclopedia, 2020. URL: https://en.wikipedia.org/wiki/Qiskit (дата обращения 10.04.2020).

[7] Welcome to Quantum / Quantum computing software, 2020. URL: https://qiskit.org/ (дата обращения 24.03.2020).

вых схем под конкретные типы вычислительных систем и версии бэкенда (локальный эмулятор или удаленные вычисления).

- Qiskit Aqua может быть использован без непосредственного квантового программирования. Данный модуль предоставляет набор инструментов для решения кросс-квантовых задач [6]. В настоящее время доступны эксперименты в области химии, искусственного интеллекта, оптимизации и финансовой сферы. Пользователь может определить какую-либо проблему и получить решение, а Qiskit Aqua реализует соответствующий квантовый алгоритм.
- Qiskit Aer предоставляет высокопроизводительный симулятор для всего стека технологий. Он содержит в себе оптимизированный C++ – симулятор для выполнения схем, скомпилированных в Qiskit Terra, а также инструменты для построения конфигурируемых моделей шума для реалистичной симуляции ошибок, возникающих во время вычислений на реальном квантовом оборудовании.
- Qiskit Ignis – это компонент, содержащий инструменты для измерения и верификации уровня шума в устройствах ближнего действия, а также позволяющий проводить вычисления в присутствии шума. Также следует отметить наличие процедур смягчения шума, которые реализованы в виде калибровочных схем и могут быть применены к наборам результатов, полученных с использованием одной и той же платформы.

Главным преимуществом Qiskit является возможность бесплатно проводить вычисления на суперкомпьютере IBM с предоставлением 5 кубитов. Для этого требуется создать аккаунт IBM Q Experience и получить токен для дальнейшего доступа. Рассмотрим пример на языке программирования Python:

```
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister, execute
from qiskit import IBMQ
from qiskit.providers.ibmq import least_busy
IBMQ.enable_account('PASTE_YOUR_API-KEY_HERE')
least_busy_device = least_busy(IBMQ.get_provider().backends(simulator=False))
q = QuantumRegister(1)
c = ClassicalRegister(1)
state = QuantumCircuit(q, c)
state.measure(q, c)
state.draw(output='mpl')
job = execute(state, backend=least_busy_device, shots=1024)
import time
while job.status().name != 'DONE':
   print(job.status())
   time.sleep(10)
result = job.result()
from qiskit.tools.visualization import plot_histogram
plot_histogram(result.get_counts(state))
```

После импорта всех необходимых сущностей активируется аккаунт с использованием полученного ранее токена, находится наиболее доступное устройство для проведения вычислений, создаются квантовый и классический регистры и соответствующая квантовая схема, на которую добавляется гейт измерения.

Команда *state.draw(output='mpl')* позволяет отобразить текущее состояние схемы. В данном случае она будет выглядеть так, как показано на рис. 1.

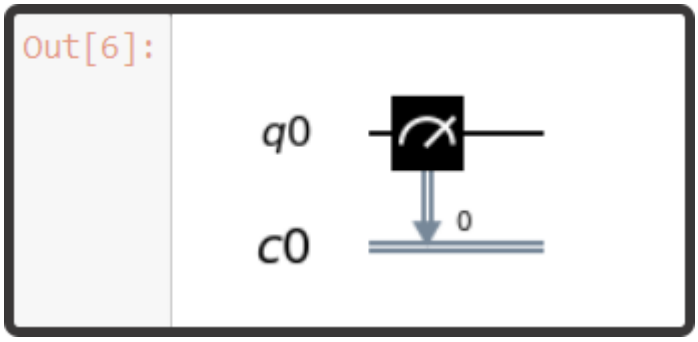

Рис. 1. Отображения состояния квантовой схемы
Fig. 1. Quantum scheme image

Далее создается задание на выполнение схемы (функция *execute*) на реальном устройстве, и с периодичностью 10 секунд проверяется статус выполнения задания. Когда задание создано, оно помещается в очередь со статусом QUEUED, с началом выполнения статус сменяется на RUNNING, а по завершении принимает значение DONE.

Визуализировать результат можно с использованием столбчатой диаграммы. На рис. 2 показан примерный вывод состояния кубита.

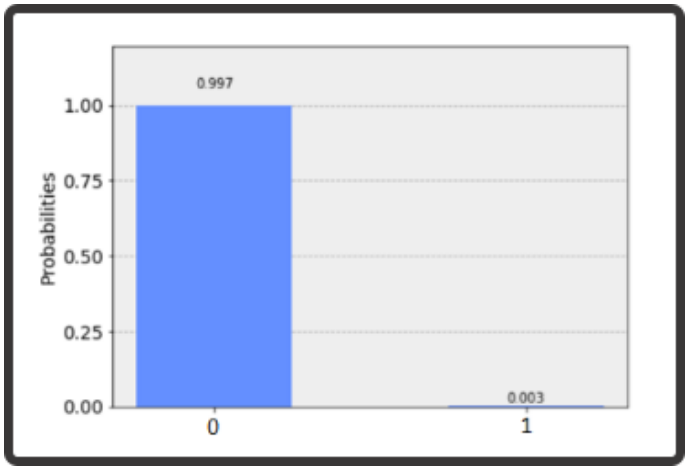

Рис. 2. Визуализация результатов
Fig. 2. Results Visualization

Таким образом, главной особенностью Qiskit, безусловно, является уникальная возможность проведения вычислений на действительно существующем квантовом устройстве, а не только с использованием локального эмулятора. Однако у данного преимущества есть обратная сторона – ждать начала выполнения задачи приходится довольно долго, в данном случае время ожидания составило порядка 9 минут.

Также достоинствами являются открытый исходный код и подробная документированность – на официальном сайте можно найти инструкции по установке фреймворка и примеры использования.

4. **Quantum Programming Studio.** Данный сервис представляет собой веб-интерфейс, который позволяет создавать квантовые схемы и алгоритмы, а после получать результаты путем непосредственного моделирования в окне браузера. При этом вычисления могут осуще-

ствляться, как в режиме эмуляции с помощью встроенного пакета Quantum cirquit (был рассмотрен ранее), так и непосредственно на квантовом компьютере [8].

Quantum Programming Studio (QPS) позволяет конструировать квантовые схемы путем простого перетаскивания ее элементов из специальной панели инструментов, что показано на рис. 3.

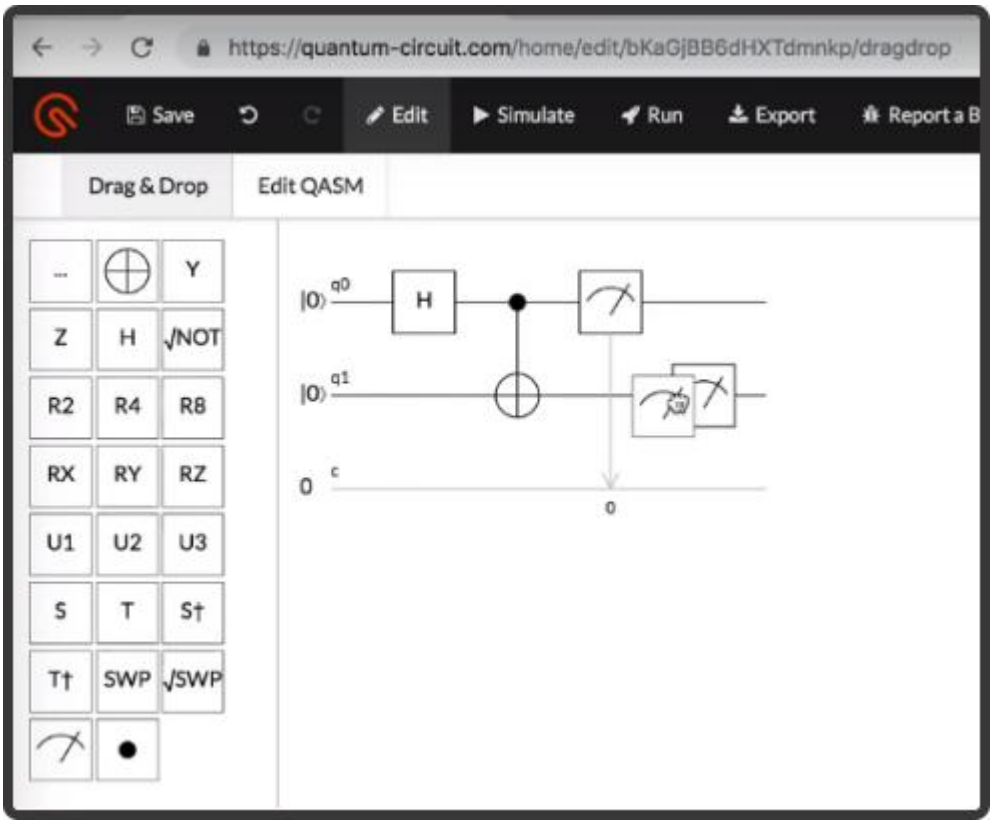

Рис. 3. Создание квантовый схемы в QPS
Fig. 3. Creating quantum circuits in QPS

Каждый клиент сервиса QPS взаимодействует лишь с его графическим веб-интерфейсом. Непосредственные вычисления могут происходить на сторонних сервисах и эмуляторах и в редких случаях на клиентской ЭВМ. QPS позволяет работать с Rigetti QCS, Rigetti Forest SDK и IBM Qiskit на выбор. Эти сервисы позволяют выполнять вычисления на своих квантовых компьютерах и специальных эмуляторах [7]. Обычно пользователь QPS взаимодействует с веб-интерфейсом локально у себя на ЭВМ посредством HTTPS протокола. Также и выполнение квантовых схем происходит либо на пользовательском компьютере, либо в облаке, где установлены все необходимые программные зависимости. Для этого в процессе работы происходит установка соединения через защищенный веб-сокет между сервером QPS и QPS-клиентом, который выполняет квантовые схемы на серверном симуляторе или же на оборудовании партнерских компаний. QPS-клиент является адаптером между запросами пользователя и непосредственно реальным оборудованием, которое предоставляют партнеры QPS. Схема полного взаимодействия пользователя с сервисом представлена на рис. 4.

[8] Quantum Programming studio, 2020. URL: https://quantum-circuit.com/docs (дата обращения 27.03.2020).

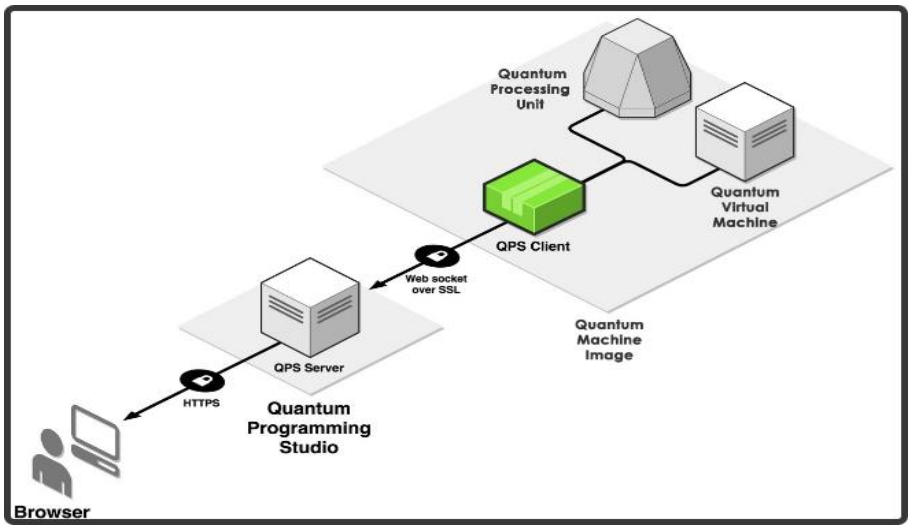

Рис. 4. Взаимодействие пользователя с сервисом QPS
Fig. 4. User Interaction with QPS

Отмечая все особенности QPS, можно выделить ряд достоинств и недостатков. К достоинствам, несомненно, можно отнести возможность выполнения вычислений в облаке в режиме эмулятора и, конечно же, тот факт, что QPS может работать с разными реализациями настоящих квантовых компьютеров. Здесь также стоит сказать о кросс-платформенности и гибкости всего сервиса QPS: пользователь сможет разрабатывать и выполнять свои квантовые решения с абсолютно любого устройства и из любой точки мира. Недостатком по сути является тот факт, что если пользователь захочет использовать локальные эмуляторы, то ему придется устанавливать их самостоятельно.

5. **Microsoft Quantum Development Kit (MQDK).** Выше были рассмотрены различные конкретные решения для работы с квантовыми алгоритмами и компьютерами, а MQDK представляет собой целую инфраструктуру, которая может пригодиться как начинающему, так и уже опытному разработчику в области квантовых технологий.

Основным инструментом для всех инструментов MQDK служит квантово-ориентированный язык программирования Q# и библиотеки для него, которые находятся в полностью открытом доступе [9]. С их помощью даже новичок сможет построить свое квантовое решение. Доступен Q# как составляющий элемент загружаемого для Visual Studio плагина MQDK. Данный язык также считается новаторским в области квантовых компьютеров, предоставляет весь инструментарий разработчика, включая полноценную отладку написанных приложений и возможность оценить затраты на запуск решения. На данный момент Q# способен в условиях эмуляции работать с 30 кубитами. Стоит отметить, что Q# совместим с языком Python. Эта совместимость по сути заключается в наличии модуля qsharp для Python, который позволяет запускать программы, написанные на языке Q# [10].

Конечно же, MQDK должен обладать возможностью использовать для вычисления квантовых решений настоящие квантовые компьютеры и специализированное высокопроизводительное программное обеспечение. С этой целью создан сервис Azure Quantum, который представляет собой огромный набор служб, включает готовые квантовые решения, программное обеспечение, способное достичь связности порядка 40 кубитов. Azure Quantum

[9] Microsoft Quantum Documentation / Microsoft Docs, 2020. URL: https://docs.microsoft.com/en-us/quantum/ (дата обращения 08.04.2020).
[10] Там же.

представляет своим пользователям доступ к наиболее конкурентно способным предложениям на рынке квантовых технологий. Данный сервис дает возможность использовать инструменты, которые можно разделить на следующие категории:

- готовые квантовые решения, работающие в промышленных масштабах;
- квантовое ПО (симуляторы, средства оценки ресурсов);
- квантовая аппаратная система с множеством конфигураций связанных кубитов;
- инструменты масштабирования, безопасности и постоянной поддержки от команды Azure и их партнеров.

Структура всей системы Azure Quantum представлена на рис. 5.

Разработчики MQDK также позаботились о теоретической и практической подготовленности своих пользователей. Благодаря коллекции учебных пособий под названием «Quantum Katas» каждый может изучить на наглядных примерах принципы квантового программирования и научиться работать со всей инфраструктурой MQDK. Также на официальном сайте MQDK можно найти ссылку на GitHub репозиторий с всевозможными рабочими примерами квантовых решений [11].

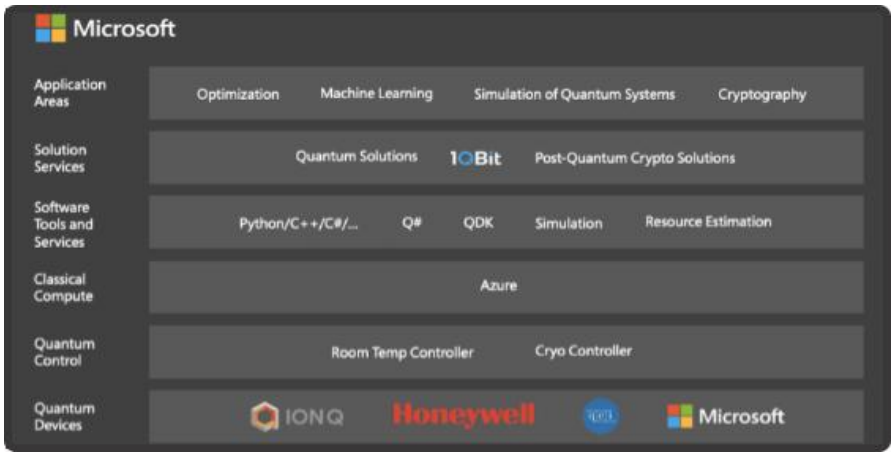

Рис. 5. Стек сервиса Azure Quantum
Fig. 5. User Interaction with QPS

В целом MQDK является наиболее оптимальным решением для разработки и изучения квантовых решений. Уже на данный момент MQDK – это набор очень удобных инструментов, которые помогут овладеть знаниями в области квантовых вычислений, научиться разрабатывать и оперировать квантовыми решениями.

Заключение. В данной работе рассмотрены различные библиотеки для проведения квантовых вычислений, проведено их сравнение и проанализированы достоинства и недостатки. Каждый из инструментов предназначен для использования с различным языком программирования, однако все они имеют открытый исходный код и распространяются бесплатно.

Возможности библиотек в целом схожи, однако Quest и Qiskit выделяются на общем фоне: первая благодаря поддержке распределенных вычислений, вторая из-за возможности использования 5 кубит квантового суперкомпьютера IBM для расчета схем. Конечно, на данный момент реализация технологии имеет недостатки, но сама возможность подключиться к квантовому компьютеру, пусть и в учебных целях, уникальна.

Выбор конкретного инструмента каждый пользователь должен осуществлять сам, в зависимости от своих целей и предпочтений. Авторам данной статьи наиболее импонирует биб-

[11] Microsoft Quantum Documentation / Microsoft Docs, 2020.

лиотека Qiskit благодаря возможности создавать квантовые схемы на высоком уровне абстракции, хорошей документированности и использования имеющего низкий порог вхождения языка Python в качестве основной платформы.

Список литературы

1. **Манин Ю.** Вычислимое и невычислимое. М.: Сов. радио, 1980. 128 с.
2. **Feynman R. P.** Simulating physics with computers. *International Journal of Theoretical Physics*, 1982, no. 21, p. 467–488. DOI 10.1007/BF02650179
3. National Academies of Sciences, Engineering, and Medicine. Quantum Computing: Progress and Prospects. Washington, DC, The National Academies Press, 2018. DOI 10.17226/25196
4. **Aryte F., Arya K., Babbush R. et al.** Quantum supremacy using a programmable superconducting processor. *Nature*, 2019, no. 574, p. 505–510. DOI 10.1038/s41586-019-1666-5
5. **Jones T., Brown A., Bush I., Benjamin S. C.** QuEST and High Performance Simulation of Quantum Computers. *Scientific Reports volume*, 2019, no. 9. DOI 10.1038/s41598-019-47174-9
6. **Moran C.** Mastering Quantum Computing with IBM QX: Explore the world of quantum computing using the Quantum Composer and Qiskit. Packt Publishing, 2019. 308 c.
7. **Kaiser S. C., Granade Ch. E.** Learn Quantum Computing with Python and Q#. Manning Publication, 2020. URL: https://www.manning.com/books/learn-quantum-computing-with-python-and-q-sharp#reviews (дата обращения 03.04.2020).

References

1. **Manin Yu.** Computable and non-computable. Moscow, Soviet Radio Publ., 1980, 128 p. (in Russ.)
2. **Feynman R. P.** Simulating physics with computers. *International Journal of Theoretical Physics*, 1982, no. 21, p. 467–488. DOI 10.1007/BF02650179
3. National Academies of Sciences, Engineering, and Medicine. Quantum Computing: Progress and Prospects. Washington, DC, The National Academies Press, 2018. DOI 10.17226/25196
4. **Aryte F., Arya K., Babbush R. et al.** Quantum supremacy using a programmable superconducting processor. *Nature*, 2019, no. 574, p. 505–510. DOI 10.1038/s41586-019-1666-5
5. **Jones T., Brown A., Bush I., Benjamin S. C.** QuEST and High Performance Simulation of Quantum Computers. *Scientific Reports volume*, 2019, no. 9. DOI 10.1038/s41598-019-47174-9
6. **Moran C.** Mastering Quantum Computing with IBM QX: Explore the world of quantum computing using the Quantum Composer and Qiskit. Packt Publishing, 2019. 308 c.
7. **Kaiser S. C., Granade Ch. E.** Learn Quantum Computing with Python and Q#. Manning Publication, 2020. URL: https://www.manning.com/books/learn-quantum-computing-with-python-and-q-sharp#reviews (дата обращения 03.04.2020).

Материал поступил в редколлегию
Received
03.05.2020

Сведения об авторах

Баскаков Павел Евгеньевич, студент 5-го курса кафедры «Кибербезопасность информационных систем» Донского государственного технического университета (Ростов-на-Дону, Россия)
pavelbaskakov98@gmail.com

Хабовец Юрий Юрьевич, студент 5-го курса кафедры «Кибербезопасность информационных систем» Донского государственного технического университета (Ростов-на-Дону, Россия)
yuriy131196@gmail.com

Пилипенко Ирина Александровна, ассистент кафедры «Кибербезопасность информационных систем» Донского государственного технического университета (Ростов-на-Дону, Россия)
irenphil@yandex.ru

Кравченко Вероника Олеговна, аспирант 3-го года кафедры «Кибербезопасность информационных систем» Донского государственного технического университета (Ростов-на-Дону, Россия)
olenikr@yandex.ru

Черкесова Лариса Владимировна, доктор технических наук, профессор кафедры «Кибербезопасность информационных систем» Донского государственного технического университета (Ростов-на-Дону, Россия)
chia2002@inbox.ru

Information about the Authors

Pavel E. Baskakov, 5th year student of the Department “Cybersecurity of information systems” of the Don State Technical University (Rostov on Don, Russian Federation)
pavelbaskakov98@gmail.com

Yuri Yu. Khabovets, 5th year student of the Department “Cybersecurity of information systems” of the Don State Technical University (Rostov on Don, Russian Federation)
yuriy131196@gmail.com

Irina A. Pilipenko, assistant of the Department “Cybersecurity of information systems” of the Don State Technical University (Rostov on Don, Russian Federation)
irenphil@yandex.ru

Veronika O. Kravchenko, 3-year post-graduate student of the Department “Cybersecurity of information systems” of the Don State Technical University (Rostov on Don, Russian Federation)
olenikr@yandex.ru

Larisa V. Cherkesova, doctor of technical Sciences, Professor of the Department “Cybersecurity of information systems” of the Don State Technical University (Rostov on Don, Russian Federation)
chia2002@inbox.ru

УДК 004.932.72’1
DOI 10.25205/1818-7900-2020-18-2-54-61

Автоматическое тегирование изображений одежды

А. Г. Малышев, А. С. Полыгалов, С. А. Алямкин

ООО «Экспасофт»
Новосибирск, Россия

Аннотация

Описывается разработка системы автоматической разметки изображений одежды человекочитаемыми атрибутами (тегами). Подобные системы становятся востребованными в сфере коммерции для пополнения информации об инвентаре и улучшения его организации, а также реализации интерактивного поиска по фотографиям, доступного клиентам. Построенное решение способно выполнять автоматический анализ атрибутов длины, дизайна и цвета для произвольного количества предметов одежды на фотографии. Архитектура решения позволяет менять набор предсказываемых тегов или переходить к решению задач тегирования на других данных.

Ключевые слова

машинное обучение, обработка изображений, компьютерное зрение, тегирование

Для цитирования

Малышев А. Г., Полыгалов А. С., Алямкин С. А. Автоматическое тегирование изображений одежды // Вестник НГУ. Серия: Информационные технологии. 2020. Т. 18, № 2. С. 54–61. DOI 10.25205/1818-7900-2020-18-2-54-61

Automatic Tagging of Clothing Images

A. G. Malyshev, A. S. Polygalov, S. A. Alyamkin

Expasoft LLC
Novosibirsk, Russian Federation

Abstract

This paper presents a computer vision clothing auto-tagging algorithm. Tagging is highly demanded in e-commerce as a tool to create a rich uniform set of annotations. The annotations improve catalog organization, statistics, and can be used for interactive catalog search by consumer photos. The proposed algorithm predicts length, design, and color attributes for an arbitrary number of clothing items in an image. The modular structure of the proposed system allows reconfiguration for other sets of tags and tagging tasks not related to clothing.

Keywords

machine learning, image processing, computer vision, tagging

For citation

Malyshev A. G., Polygalov A. S., Alyamkin S. A. Automatic Tagging of Clothing Images. *Vestnik NSU. Series: Information Technologies*, 2020, vol. 18, no. 2, p. 54–61. (in Russ.) DOI 10.25205/1818-7900-2020-18-2-54-61

Введение

Постоянно меняющиеся коллекции одежды, растущее разнообразие стилей и большой ассортимент существенно усложняют как задачу выбора товара для покупателя, так и задачу систематизации инвентаря и рекомендации подходящего предмета для продавца.

Основной способ борьбы с данной проблемой – составление каталогов с объединением товаров в иерархию категорий, в которых проще ориентироваться. Однако разнообразие товаров уже стало настолько высоким, что внутри привычных категорий, таких как «футболка», «водолазка», «толстовка», нужно дополнительное структурирование для эффективного поиска подходящих вариантов.

Дополнительную информацию о товарах могут предоставлять производители, но разные производители часто предлагают несовпадающие наборы атрибутов: для футболок, например, производитель А может описать цвет и посадку, а производитель Б – цвет и рисунок. В таком случае без дополнительной разметки атрибуты «посадка» и «рисунок» не позволяют работать со всей базой, и нужен способ восполнить пробелы. Кроме того, многих атрибутов, способных помочь при поиске, обычно вообще нет, или они недостаточно детальны. Дополнительное преимущество автоматической разметки над ручной – возможность предоставить ее покупателю для автоматического поиска одежды по фотографии.

Для разметки товаров можно нанимать людей, но при большом регулярно обновляющемся инвентаре временные и материальные расходы на такое решение относительно высоки, поэтому в сфере коммерции растет интерес к автоматическим методам разметки.

В данной работе предлагается автоматическая система разметки одежды человекочитаемыми атрибутами (тегами) длины, дизайна и цвета, способная обрабатывать несколько предметов одежды на изображении. Приводятся особенности разработки каждого компонента алгоритма и работы с данными. Архитектура итогового решения обеспечивает обработку произвольного количества предметов одежды на изображении и предусматривает изменения в наборе предсказываемых тегов, а также возможность применения к задачам тегирования на других данных.

Методы анализа изображений

Методы анализа изображений принято делить на классические и нейросетевые, распространившиеся позже. Классические методы основываются на поиске характерных элементов на изображении и вычислении статистик по заранее заданному вручную алгоритму. Основная проблема в разработке и применении этого семейства методов – высокая сложность разработки новых алгоритмов или настройки существующих для решения новой задачи: каждый случай требует сложного алгоритмического описания и индивидуальной настройки параметров. Нейросетевые методы выигрывают у классических возможностью одновременной оптимизации извлечения наиболее эффективных признаков и их обработки. Это обеспечило их лидирующие позиции в задачах анализа изображений, таких как классификация, сегментация и детектирование. В связи с этим в данной работе предпочтение отдано нейросетевым алгоритмам.

Для обработки нескольких объектов на изображении необходимо их обнаруживать и различать, что является задачей детектирования – определения регионов, где есть интересующие предметы. Для экспериментов выбрана библиотека Tensorflow Object Detection API [1] с использованием алгоритма Single-Shot Detector [2] и архитектуры MobileNet v2 [3] в силу простоты использования, наличия предобученных моделей и высокой производительности.

Сама задача тегирования в данной работе решалась как задача классификации. Для экспериментов выбраны классификаторы на основе архитектур Inception v4 [4] и MobileNet v2. Данные архитектуры показали лучшие для своего размера результаты по качеству классификации и позволили проверить зависимость качества от размера сети.

Для задачи распознавания цвета нужно дополнительно определять, какие пикселы принадлежат интересующему предмету – для этого решается задача семантической сегментации. Для экспериментов в силу лучшего на момент обзора качества и высокой производительности выбран алгоритм DeepLab v3+ [5].

Описание решения

В результате проектирования и экспериментов была установлена схема обработки изображений, показанная на рисунке. Изображение сначала проходит процедуру детектирования для устранения лишних деталей и нормализации масштабов, затем выделенная область проходит через алгоритм сегментации, и в итоге по региону и маске происходит предсказание атрибутов длины, дизайна и цвета. Преимущества этого подхода в возможности обработки произвольного количества объектов на изображении и дополнения или замены модулей классификации. Это позволяет дополнять систему тегирования новыми классификаторами по мере их разработки или переходить к решению задачи тегирования на других данных.

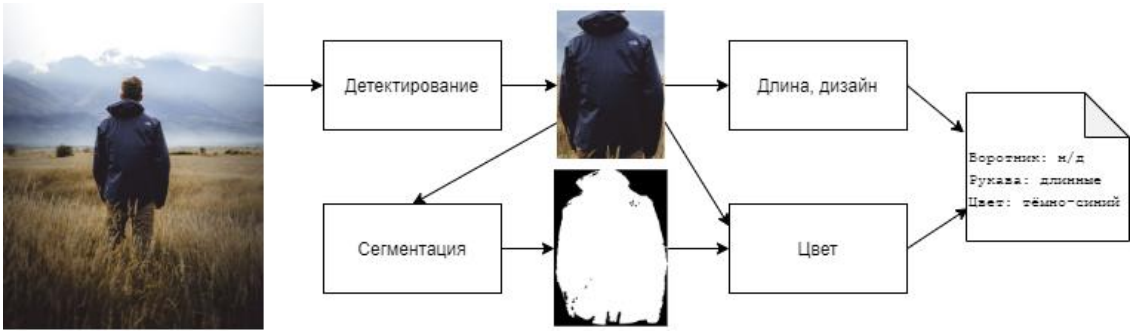

Схема обработки изображений
Image processing sequence

Детектирование

Для обучения и тестирования алгоритма детектирования использовалась подвыборка из коллекций DeepFashion [6] и ModaNet [7]. Выделенный набор данных был вручную провалидирован в целях исключения ошибок разметки. Также были исключены изображения с низкой контрастностью, слишком близким ракурсом, с сильным пересечением предметов и предметами, частично выходящими за кадр. Очистка производилась полуавтоматическим способом: на основе аннотаций изображений строились фильтрующие правила, и корректность отсечения проверялась вручную. Полученные данные балансировались по категории.

В коллекции DeepFashion присутствуют аннотации обрамляющих прямоугольников (bounding boxes), ключевых точек (landmarks), категории одежды и прочих атрибутов, не используемых в этой работе.

При детальном изучении выяснилось, что обрамляющие прямоугольники построены по ключевым точкам, что позволяет размножать данные поворотом и перспективными искажениями и генерировать по ним новые корректные прямоугольники.

Много полезной информации удалось извлечь из ключевых точек. Ключевые точки описывают положение частей одежды в кадре: например, концов рукавов, плеч, подола. Каждая точка описана ее координатами на изображении и флагом видимости. Если точка не видна, но ее положение можно угадать по окружающим частям одежды, она указывается как неви-

димая, но с заполненными координатами. Если точку указать невозможно – она не применима к данному виду одежды, как положение молнии на футболке, или не попала в кадр, ее координата не заполняется. Используя положения, видимость и заполненность точек на многих изображениях удалось пополнить имеющуюся информацию и проверить качество съемки. По длине прилегания области, содержащей объект, к краю изображения установлено, на каких изображениях съемка произведена слишком близко, и объект не полностью попадает в кадр. По пересечению областей, содержащих предметы, установлено, на каких фотографиях два человека стоят друг за другом. По видимости и заполненности ключевых точек воротника получена информация, спереди снят объект или сзади. Если на человеке размечено несколько предметов одежды, они расположены один над другим, и информацию о ракурсе можно распространить на них. Также убраны слишком мелкие по площади предметы. Наконец, отсеяны темные и низкоконтрастные фотографии, где слишком низкая средняя интенсивность и много шума или малый разброс значений.

Коллекция ModaNet содержит разметку областей, где находятся предметы, в виде попиксельных масок, что позволяет использовать эти данные для обучения как детектированию, так и сегментации. К этой коллекции также применимо размножение с помощью пространственных искажений. Для того чтобы эту коллекцию объединить с DeepFashion, произведена переразметка категорий одежды и убраны неактуальные категории, такие как шарфы, сумки и обувь. Как и в случае с DeepFashion, отсеяны слишком крупные и мелкие предметы.

Коллекции объединены и сбалансированы по категориям и положению объектов в кадре, чтобы избежать переобучения под конкретный стиль фотографии. Итоговый результат сохранен как последовательность изображений в бинарном формате TF Records, требуемом со стороны TF OD API. Балансировка происходила посредством равновероятного семплирования из каждой категории с размножением изображений из маленьких подмножеств. В частности, данный подход к организации данных гарантирует разнообразие изображений на каждом шаге обучения, что ускоряет сходимость алгоритма и улучшает качество итоговой модели.

Обучение производилось с применением метода transfer learning [8] с модели, обученной для детектирования на коллекции MS COCO [9], с поиском сложных примеров с высокой уверенностью ошибочного предсказания (> 0,99) и размножением данных в процессе обучения случайной обрезкой и отражениями по горизонтали.

Классификация атрибутов длины и дизайна

На примере атрибутов длины и дизайна изучены особенности обработки визуально близких атрибутов с упорядоченностью и без нее. Для обучения применялась коллекция FashionAI [1], состоящая из студийных фотографий, на каждой из которых позирует один человек в различных окружениях. На каждой фотографии размечен ровно один атрибут длины или дизайна элемента одежды. Разметка выполнена в формате «да / нет / наверное» для всех атрибутов; для случаев, когда атрибут не применим или соответствующая ему часть одежды не видна, предусмотрено значение атрибута «Invisible».

Разметка была переведена в более строгий формат: оставлены только объекты, где однозначно определяется атрибут (ровно один ответ «да» либо ровно один ответ «наверное» при отсутствии «да»). Результат проверен вручную. Выяснилось, что значение «Invisible» для атрибутов длины использовалось очень неаккуратно: как правило, оно не означало, что длину определить нельзя, и фотографии с ним ничем не отличались от фотографий с прочими значениями атрибута длины. Для лучшей интерпретируемости результатов такие образцы были убраны из коллекции.

[1] tianchi.allyun.com. Fashionai global challenge. URL: https://tianchi.aliyun.com/competition/rankingList.htm?spm=5176.11409106.5678.4.6510d751R1Fce0&raceId=231649.

Обучение производилось с предобученных на ImageNet [10] моделей с настройкой параметров обучения по процедуре, описанной в [11]. Для определения оптимальной стратегии обработки данных проведена серия экспериментов. Сравнивались модели, предсказывающие атрибуты длины и дизайна вместе и по отдельности. За счет присутствия в кадре только одного человека удалось проверить влияние обрезки изображения вокруг интересующего предмета одежды по обрамляющему прямоугольнику. Результаты показывают, что выгодно производить предсказания длины и дизайна в отдельных нейросетях и что обрезка имеет положительное влияние на качество предсказаний – предположительно, из-за отсечения лишних деталей на фоне. Также установлено, что влияние качественных изменений (обрезка) превосходит выгоду от перехода к многократно более тяжеловесной архитектуре. Дополнительно обнаружено, что при предсказании визуально близких упорядоченных атрибутов (длины) в большинстве случаев ошибка приводит к предсказанию соседнего класса, что позволяет для задачи поиска по каталогу объединить соседние классы и получить выдачу высокой точности.

Сегментация

Сегментация в данной работе является вспомогательным компонентом для задачи определения цвета. Получаемые маски используются для сбора пикселов, принадлежащих интересующему предмету.

Задача семантической сегментации обычно решается как задача попиксельной классификации для определения, какие пикселы принадлежат фону, а какие – предмету. У данного подхода есть недостаток, заключающийся в том, что на самом деле нас интересует не просто точность классификации, а точность соответствия предсказанной маски эталонной: как хорошо воспроизводится форма предмета, есть ли в предсказанной маске пропуски, выходит ли она за границы эталона. Численно это выражается мерой Жаккара – отношением площади пересечения к площади объединения, обладающей инвариантностью к масштабу и хорошо согласующейся с восприятием качества локализации. Для непосредственной оптимизации меры Жаккара предложена функция потерь [имени] Ловаса [12]. Для экспериментов данная функция потерь была добавлена в библиотеку для DeepLab v3+. Итоговая функция потерь – взвешенная сумма многоклассовой кросс-энтропии и функции Ловаса. Для обучения и тестирования использована коллекция ModaNet с той же предобработкой, что для задачи детектирования.

Анализ предсказаний базового решения с «наивной» предобработкой данных (обрезка ровно по обрамляющему прямоугольнику с приведением к размеру входа нейросети) показал несколько характерных особенностей в ошибках предсказания: часто происходит предсказание семантически близкого класса (например, куртка вместо футболки); контрастные элементы могут предсказываться как фон; большинство ошибок в определении края объекта случается около границ изображения.

Первым потенциальным улучшением стала проверка того, как влияет сохранение соотношения сторон на качество предсказаний. В базовом решении эта информация терялась из-за обрезки ровно по прямоугольнику. Для устранения этого прямоугольники увеличиваются до квадрата – таким образом, предметы любой формы могут быть поданы на вход нейросети без искажений. Из-за увеличения обрамляющего прямоугольника возникает два дополнительных вопроса. 1. Где расположить предмет внутри нового прямоугольника? Рассмотрены два варианта: по центру и в случайном месте. 2. Что делать, если часть прямоугольника не помещается на изображении: обрезать не поместившиеся части, закрасить черным, заполнить значениями на краю изображения или заполнить отражением? Лучше всего сработало центрирование и заполнение отражением.

Следующим проверено предположение об улучшении предсказаний за счет увеличения обрамляющего прямоугольника для обеспечения дополнительного контекста около границ. Увеличение происходило на 10, 25 и 50 %, лучше всего сработало увеличение на 25 %.

Далее проверено влияние добавления функции потерь Ловаса к кросс-энтропии. Лучше всего сработало сложение кросс-энтропии и функции Ловаса с весом 0,5.

Также проверено использование поиска сложных пикселов для акцентирования обучения на областях, где предсказание происходит наименее надежно. При использовании этой процедуры градиент подсчитывался по 1, 10, 50 % изображения с наивысшим значением функции потерь. Ни один из вариантов не дал улучшения относительно обучения без применения поиска, поэтому в финальном варианте поиск не используется.

Некоторые ошибки, такие как небольшие пробелы внутри масок или ошибки классификации, достаточно просто описываются и стабильно воспроизводятся, чтобы можно было их исправить алгоритмически. Небольшие пробелы в масках и ложные срабатывания устраняются путем переразметки аномальной области в класс окружающего предмета или фона. Ошибки классификации устраняются на основе знания ожидаемого класса объекта, предсказываемого детектором, характерных ошибок (футболки путаются с куртками, но не со штанами) и характерного распределения площадей для каждого ожидаемого класса: перекрашивая области, наиболее выбивающиеся из распределения, в целевой класс, можно понять, какая из областей на изображении была классифицирована неправильно.

Предсказание цвета

Задача предсказания цвета в данной работе решается как задача классификации. Основная сложность состоит в том, что работа происходит с малой коллекцией изображений с большим количеством оттенков: изображения распределены приблизительно по 1 000 оттенков, которые объединены в 100 промежуточных и 20 основных цветов. Прямое решение задачи классификации при этом дает неудовлетворительный результат: много ошибок, причем при ошибках часто предсказывается совершенно непохожий цвет. Для исправления этой проблемы решено проверить методы обучения распознавания сходства, распространенные в задаче распознавания лиц. Задача распознавания лиц обладает схожими проблемами: небольшое количество данных, высокое количество классов (уникальных лиц), высокие требования к качеству предсказаний. Обучение распознаванию сходства заставляет нейросеть, во-первых, располагать схожие объекты ближе в пространстве признаков, что должно решить проблему с предсказанием непохожих цветов в случае ошибки, а также улучшить надежность предсказаний в целом. Для тестирования выбраны показывающие лучшие результаты методы Triplet loss [13] и ArcFace [14]. Принцип работы обоих методов – группировать похожие объекты и обеспечивать настраиваемую минимальную дистанцию до непохожих, что повышает устойчивость к вариациям во входных данных и помогает в задаче классификации.

В базовом решении обнаружено, что цвет фона может влиять на предсказание. Для улучшения качества предсказаний протестировано применение масок сегментации и аугментации фона. Маски либо добавлялись к изображению четвертым каналом, либо использовались для закраски всех пикселов, не относящихся к изображению, в черный цвет. Лучше сработал первый вариант. В качестве аугментации применялась замена фона на случайное изображение, геометрические искажения и искажения цвета. Положительный эффект дали все аугментации, кроме искажения цвета.

Поскольку разметка цвета иерархическая (есть три уровня подробности разбиения), протестированы подходы к обучению модели предсказывать цвета по более мелкому разбиению и затем предсказывать по более грубому, а также к обучению модели предсказывать цвета сразу на нескольких уровнях подробности. Лучше сработал последний вариант, когда модель одновременно учится упорядочивать раскраски по схожести по самому мелкому разбиению и предсказывать цвета по мелкому, среднему и крупному разбиениям.

В дополнение к Triplet loss протестирован декодирующий модуль для предсказания цвета как абсолютной величины для обеспечения дополнительной информации во время обучения. Положительного влияния на качество предсказаний этим методом добиться не удалось.

Также в дополнение к Triplet loss протестировано использование декодирующих модулей, реализующих Arcface. Наилучший результат получен при замене всех модулей классификации на Arcface и сохранении Triplet loss в качестве ограничения на векторы признаков.

Заключение

Разработана и реализована расширяемая архитектура системы тегирования. Полученная система позволяет обрабатывать изображения одежды и обеспечивать поиск по каталогу по фотографии.

Рассмотрены особенности реализации каждого этапа обработки изображения. Исследованы особенности подготовки данных и работы с визуально близкими упорядоченными и неупорядоченными атрибутами, влияние локализации на качество классификации, приемы улучшения качества классификации в случае большого количества классов.

Полученная архитектура, как и приемы предобработки изображений, настройки параметров обучения и анализа предсказаний, универсальна и может применяться в будущих системах, решающих смежные задачи на других данных.

Список литературы / References

1. **Huang J., Rathod V., Sun C. et al.** Speed / accuracy trade-offs for modern convolutional object detectors. In: arXiv:1611.10012, 2016.
2. **Wei Liu, Dragomir Anguelov, Dumitru Erhan et al.** Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, 2016, p. 21–37.
3. **Sandler M., Howard A., Zhu M. et al.** Mobilenetv2: Inverted residuals and linear bottlenecks. In: arXiv:1801.04381, 2018.
4. **Szegedy C., Ioffe S., Vanhoucke V., Alemi A.** Inception-v4, inceptionresnet and the impact of residual connections on learning. In: arXiv:1602.07261, 2016.
5. **Chen L.-C., Zhu Y., Papandreou G. et al.** Encoder-decoder with atrous separable convolution for semantic image segmentation. In: arXiv:1802.02611, 2018.
6. **Z. Liu, P. Luo, S. Qiu et al.** Deepfashion: Powering robust clothes recognition and retrieval with rich annotations. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, p. 1096–1104.
7. **Zheng S., Yang F., Kiapour M. H., Piramuthu R**. Modanet: A large-scale street fashion dataset with polygon annotations. In: arXiv:1807.01394, 2018.
8. **Tan C., Sun F., Kong T. et al.** A survey on deep transfer learning. In: arXiv:1808.01974, 2018.
9. **Tsung-Yi Lin, Michael Maire, Serge Belongie et al.** Microsoft coco: Common objects in context. In: Computer Vision – ECCV 2014. Eds. David Fleet, Tomas Pajdla, Bernt Schiele, Tinne Tuytelaars. Cham, Springer International Publishing, 2014, p. 740–755.
10. **Russakovsky O., Deng J., Su H. et al.** Imagenet large scale visual recognition challenge. In: arXiv:1409.0575, 2014.
11. **Page D.** On hyperparameter tuning and how to avoid it. URL: https://myrtle.ai/ how-to-train-your-resnet-5-hyperparameters/.
12. **Berman M., Triki A. R., Blaschko M. B.** The lov´asz-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In: arXiv:1705.08790, 2017.

13. **Schroff F., Kalenichenko D., Philbin J.** Facenet: A unified embedding for face recognition and clustering. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, Jun.
14. **Deng J., Guo J., Xue N., Zafeiriou S.** Arcface: Additive angular margin loss for deep face recognition. In: arXiv:1801.07698, 2018.

Материал поступил в редколлегию
Received
25.06.2020

Сведения об авторах

Малышев Александр Григорьевич, разработчик, ООО «Экспасофт» (Новосибирск, Россия)
a.malyshev@expasoft.tech

Полыгалов Александр Сергеевич, ведущий разработчик, ООО «Экспасофт» (Новосибирск, Россия)
a.polygalov@expasoft.tech

Алямкин Сергей Анатольевич, технический директор, ООО «Экспасофт» (Новосибирск, Россия)
s.alyamkin@expasoft.com

Information about the Authors

Alexander G. Malyshev, developer, Expasoft LLC (Novosibirsk, Russian Federation)
a.malyshev@expasoft.tech

Alexander S. Polygalov, lead developer, Expasoft LLC (Novosibirsk, Russian Federation)
a.polygalov@expasoft.tech

Sergey A. Alyamkin, CTO, Expasoft LLC (Novosibirsk, Russian Federation)
s.alyamkin@expasoft.com

УДК 004.423.4
DOI 10.25205/1818-7900-2020-18-2-62-75

Разработка программных средств для улучшения работы механизма автодополнения кода с использованием алгоритмов машинного обучения в интегрированной среде разработки для языка Python

А. О. Матвеев [1], **А. В. Быстров** [2], **В. И. Бибаев** [3], **Н. И. Поваров** [3]

[1] *Новосибирский государственный университет*
Новосибирск, Россия

[2] *Институт систем информатики им. А. П. Ершова СО РАН*
Новосибирск, Россия

[3] *ООО «Интеллиджей Лабс»*
Санкт-Петербург, Россия

Аннотация

Автоматическое дополнение кода является важной функцией интегрированной среды разработки. Оно позволяет пользователям упростить набор длинных выражений в процессе программирования. Методы автоматического дополнения кода, как рассматриваемые в научных работах, так и реализованные в коммерческих продуктах, варьируются от применения эвристик для конкретных случаев до машинного обучения. При этом такие методы обычно опираются на статистические данные и не учитывают поведение пользователей. В статье предлагается подход к улучшению механизма автоматического дополнения кода для языка Python на основе сбора данных о работе этого механизма у реальных пользователей. Эти данные используются для обучения модели с целью последующего ранжирования вариантов автодополнения с помощью алгоритмов машинного обучения. Для обучения модели используются два типа признаков: контекстные и элементные. Контекстные признаки описывают информацию о коде рядом с позицией курсора в текстовом редакторе. Элементные признаки описывают характеристики предлагаемого варианта дополнения кода, например длину совпадающего префикса или тип варианта. Отмечается зависимость модели от ограничений на время ее срабатывания и размер. В работе также рассматриваются различные подходы к оценке качества полученной модели.

Ключевые слова

Python, PyCharm, автоматическое дополнение кода, машинное обучение, интегрированная среда разработки

Для цитирования

Матвеев А. О., *Быстров А. В.*, *Бибаев В. И.*, *Поваров Н. И.* Разработка программных средств для улучшения работы механизма автодополнения кода с использованием алгоритмов машинного обучения в интегрированной среде разработки для языка Python // Вестник НГУ. Серия: Информационные технологии. 2020. Т. 18, № 2. С. 62–75. DOI 10.25205/1818-7900-2020-18-2-62-75

Development of Software Tools to Improve the Work of the Code Completion Mechanism Using Machine Learning Algorithms in an Integrated Development Environment for Python

A. O. Matveev [1], **A. V. Bystrov** [2], **V. I. Bibaev** [3], **N. I. Povarov** [3]

[1] *Novosibirsk State University*
Novosibirsk, Russian Federation

[2] *A. P. Ershov Institute of Informatics Systems SB RAS*
Novosibirsk, Russian Federation

[3] *Intellij Labs LLC*
St. Petersburg, Russian Federation

Annotation

Auto-completion is an essential feature of any popular text editor for some language. It allows users to avoid the process of annoying typing of long expressions in their projects. There are a lot of different works in this direction in scientific research and commercial products. These works are very different and either use some special features and heuristics to improve code completion or use machine learning techniques. Most of these approaches rely on synthetic data and do not take into account the behavior of real users. The article proposes an approach to improve the automatic code completion mechanism for the Python language by collecting information about usage of this mechanism by real users. The obtained data is used to train the model to rank completion variants with machine learning algorithms. To train the model, two types of features are used: contextual and elemental. Contextual features describe information about the code next to the cursor position in a text editor. Elemental features describe the characteristics of the proposed variant, for example, the length of the matching prefix. When building a model, it is important to take into account the limits of the response time of the model and its size. Also, in the paper, various approaches of assessing the quality of the final model are considered.

Keywords

Python, PyCharm, code completion, machine learning, IDE

For citation

Matveev A. O., Bystrov A. V., Bibaev V. I., Povarov N. I. Development of Software Tools to Improve the Work of the Code Completion Mechanism Using Machine Learning Algorithms in an Integrated Development Environment for Python. *Vestnik NSU. Series: Information Technologies*, 2020, vol. 18, no. 2, p. 62–75. (in Russ.) DOI 10.25205/1818-7900-2020-18-2-62-75

Введение

Механизм автоматического дополнения кода в современных интегрированных средах разработки – один из наиболее часто используемых разработчиками, вплоть до нескольких раз в минуту [1]. Основная цель этого механизма – ускорить и упростить процесс написания программы разработчиком. Дополнительной его целью является минимизация количества опечаток и действий пользователя в целом. Автоматическое дополнение кода позволяет избежать утомительного набора длинных выражений (имен переменных, методов) в программах, исключая возможность опечатки. Кроме того, разработчики нередко не помнят точное имя метода или переменной, которую они хотят использовать. В этом случае механизм автоматического дополнения кода может предложить пользователю существующие варианты подходящего имени из списка. Еще одна полезная особенность заключается в том, что автоматическое дополнение кода побуждает разработчиков использовать более длинные и описательные имена методов и переменных, что приводит к более читаемому и понятному коду. Ввод длинных имен вручную может быть неудобен, но автоматическое дополнение кода ускоряет набор, позволяя не печатать все имя переменной или метода до конца, а выбирать из списка необходимый вариант после набора лишь части имени.

Обычно функция автоматического дополнения кода реализована в виде всплывающего окна (рис. 1), появляющегося, когда пользователь начинает вводить текст или же вызывает

эту функцию явно, с помощью комбинаций клавиш клавиатуры. Это всплывающее окно содержит возможные варианты дополнения для текущего положения в коде, например допустимые имена методов, переменных, ключевые слова, имена классов и др. Существуют реализации автоматического дополнения, в которых пользователю предлагается дополнить не одно, а сразу несколько слов до конца текущей строки или же название метода целиком. Выбранный пользователем вариант вставляется в текст. Таким образом, пользователь пишет код, не вводя весь текст.

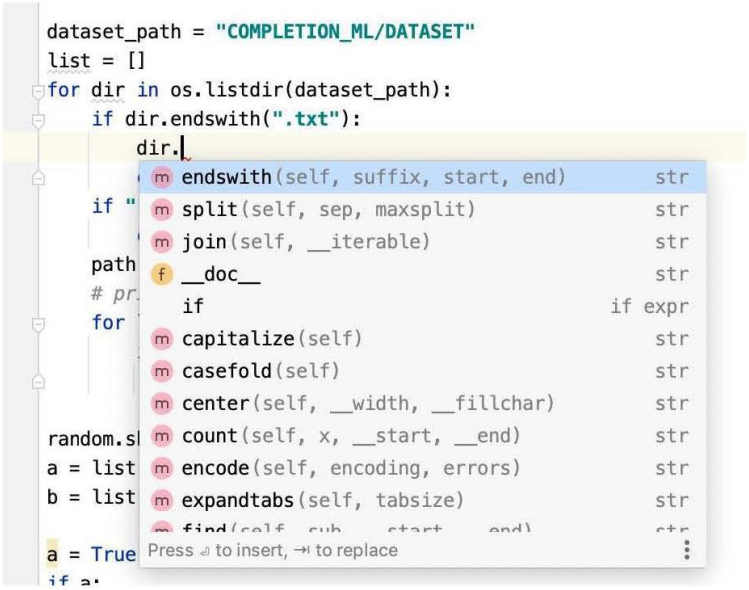

Рис. 1. Автоматическое дополнение кода в PyCharm
Fig. 1. Automatically adding code to PyCharm

Базовые системы автоматического дополнения кода весьма ограничены. В списке предлагаемых ими вариантов часто встречаются ненужные и мало используемые методы (включая методы, унаследованные от суперклассов, находящихся высоко в иерархии наследования). Это особенно часто происходит при работе с развитыми классами с большим количеством функциональных возможностей, которые можно использовать по-разному.

В настоящее время механизмы автоматического дополнения кода для интегрированных сред активно разрабатываются, появляются различные встраиваемые инструменты для популярных сред, такие как TabNine [1], IntelliCode [2], Codota [3], Kite [4] и др. В этих инструментах применяются различные подходы как к интерфейсу пользователя, так и к способам подбора вариантов. Некоторые предлагают варианты дополнения или дополняют существующий список, а есть и те, которые только сортируют уже существующие варианты. Вариантом может быть как одно слово, так и строка целиком или даже весь метод. Подходы могут отличаться и использующимися в них алгоритмами, начиная от простого набора эвристик для конкретных случаев до использования алгоритмов машинного обучения. В последнее время алгоритмы машинного обучения набирают популярность, так как они позволяют решать столь

[1] https://www.tabnine.com/.
[2] https://visualstudio.microsoft.com/ru/services/intellicode/.
[3] https://www.codota.com/.
[4] https://kite.com/.

непростую задачу в автоматическом режиме, используя готовые алгоритмы для оптимизации.

В данной статье предлагается улучшение автоматического дополнения кода для языка Python с использованием алгоритмов машинного обучения для сортировки вариантов, предлагаемых стандартными механизмами интегрированной среды для разработки PyCharm.

Академические работы в области автоматического дополнения кода

Ранние исследования в области автоматического дополнения кода были посвящены улучшению результатов работы механизма для языка Java на основе набора эвристик (Hou & Pletcher (2011) [2], Han et al. (2009) [3]). В качестве эвристик авторы использовали группировку вариантов, сортировку, основанную на иерархии типов и популярности предлагаемых вариантов, фильтрацию предлагаемых вариантов, например приватных методов.

Некоторые исследователи строили статистические модели для ранжирования и предложения вариантов автоматического дополнения кода на основе открытых источников данных. Так, Bruch и соавторы (2009) [4] предлагали искать подходящие варианты на основе существующей кодовой базы, а Robbes & Lanza (2008) [5] – на истории изменений в системах контроля версий.

Существуют также работы, базирующиеся на различных языковых моделях. Raychev и соавторы (2014) [6] используют статистические языковые модели, применяемые при обработке естественного языка, чтобы предлагать следующее слово по текстовой последовательности. Статистические модели широко используются в различных задачах обработки естественного языка. Одной из самых простых таких моделей является n-граммная модель [7], где частоты последовательных n токенов используются для прогнозирования вероятности появления последовательности токенов. Частоты таких n-грамм получены из корпуса текста и сглаживаются различными алгоритмами, такими как сглаживание Кнезера – Нея [8], чтобы учесть разреженность n-грамм. Эти модели могут быть применены и в алгоритмах автоматического дополнения кода, где каждый токен считается словом и выполняется n-граммный анализ на основе программного кода. Однако такие модели не очень хорошо фиксируют большую долю неизвестных токенов из имен переменных и функций [6].

Недавние исследования в этой области посвящены использованию моделей глубокого обучения для предложения и ранжирования вариантов автоматического дополнения кода, например Li и соавторы (2018) [9].

Коммерческие продукты

Помимо академических работ, посвященных задаче автоматического дополнения кода, существуют и коммерческие программные инструменты, предлагающие решение этой задачи. К таким инструментам, которые обычно могут быть установлены в различные интегрированные среды разработки, относятся, например, TabNine, IntelliCode, Codota, Kite. Приведем их краткое описание.

Codota – инструмент, использующий эвристики для автоматического дополнения кода всей строки. При этом в предложенных вариантах находятся в том числе и вызовы конструкторов с параметрами – локальными переменными из текущего метода, если они подходят по типу или по имени. Последовательность вызовов конструкторов или методов определяется по контексту. При этом учитывается, какие вызовы были сделаны до этого. Для языка Java имеется возможность получить информацию по использующимся методам – посмотреть похожие цепочки вызовов на github и stackoverflow.

Kite – встраиваемый в интегрированные среды разработки механизм улучшения автоматического дополнения кода для языка Python. Как и codota, учитывает предыдущие вызовы методов из контекста и предлагает подходящий вариант, основываясь на похожих примерах

из большого количества исходных кодов. Имеет функцию подстановки в аргументы функции локальных переменных (сопоставляет подходящие по имени переменные).

TabNine – инструмент для автоматического дополнения кода, работающий на основе обученной модели GPT-2 [5] (нейронная сеть, предназначенная для решения задач машинного обучения в направлении естественных языков). Модель была обучена на большом объеме исходного кода для разных языков и умеет предлагать некоторые нетривиальные варианты, используя все преимущества GPT-2. Модель позволяет по контексту определить, что нужно использовать антоним к некоторому слову, использованному ранее в контексте (например, yes – no). К недостаткам этого инструмента можно отнести большой разброс в качестве предложенных вариантов. Есть случаи, когда результаты работы весьма впечатляющи, но есть такие, в которых предложенные варианты очень далеки от того, что требуется в контексте. При этом частое появление неподходящих результатов может создавать серьезный дискомфорт для пользователя. Модель также получается слишком громоздкой и ресурсоемкой. Чтобы работать локально на устройстве пользователя, для ее работы необходимо иметь доступ к вычислительному серверу, что требует от компаний, приобретающих этот инструмент, разворачивания дополнительных систем и поддержки их в рабочем состоянии.

Описание работы механизма автоматического дополнения кода в среде PyCharm

В работе механизма автоматического дополнения кода можно выделить два этапа. Сначала в окружающих файлах и стандарте языка необходимо найти подходящие элементы и добавить в список в качестве вариантов дополнения. Затем этот список необходимо отсортировать таким образом, чтобы максимально облегчить пользователю выбор нужного варианта.

Для добавления элементов используется абстракция completion contributor [6]. Наивной реализацией данной абстракции является добавление всех элементов из текущего файла или каталога, которые соответствуют введенному пользователем префиксу. Однако для получения более качественного списка вариантов применяется множество различных completion contributor, использующих более сложные правила, основанные на анализе абстрактного синтаксического дерева, информации о типах и других факторах.

Каждый completion contributor отвечает за добавление определенного набора элементов в список. Есть отдельный completion contributor для ключевых слов, имен методов, локальных переменных. Completion contributor начинает свою работу, как только произойдет вызов автоматического дополнения кода, инициированный пользователем или автоматически. Эргономика интерфейса пользователя требует, чтобы список вариантов появился на экране не позднее, чем через 300 миллисекунд после того, как автоматическое дополнение кода инициировано. При этом может получиться так, что часть contributor-ов еще не успели закончить работу. Тогда первые 5 элементов списка, полученного к этому моменту, фиксируются и показываются пользователю сразу, а новые элементы затем добавляются в фоновом режиме в конец списка. Это необходимо для того, чтобы не получилось, что пользователь выбрал некоторый вариант, а в результате подставился другой, только что добавленный в начало списка. Таким образом, после первого показа пользователю сортировка не должна затрагивать первые 5 вариантов, а должна упорядочивать только последующие. Если же все completion contributor-ы успели отработать до первого показа пользователю вариантов, то можно сортировать весь список.

Простейшее решение для сортировки выбранных вариантов – это сортировка по алфавиту, и иногда оно работает хорошо, но когда вариантов много, пользователю приходится перемещаться по длинному списку упорядоченных по алфавиту элементов или вводить остав-

[5] https://openai.com/blog/gpt-2-1-5b-release/
[6] https://www.jetbrains.org/intellij/sdk/docs/tutorials/custom_language_support/completion_contributor.html

шуюся часть слова вручную. Поэтому нужно сортировать элементы таким образом, чтобы желаемый элемент оказался в первой позиции или как можно выше в списке. Для подбора и сортировки вариантов интегрированные среды разработки обычно используют набор эвристик. Однако эвристики слишком сложны в обслуживании и не всегда работают достаточно хорошо. Помимо этого, они плохо масштабируемы – добавление новых правил может влиять на работу существующих и не учитывать скрытые закономерности.

В среде PyCharm сортировка элементов осуществляется стандартным сортировщиком, работа которого основана на использовании упорядоченного набора так называемых completion weigher-ов. Completion weigher [7] – это абстракция, задача которой – задать вес каждому элементу. Сначала элементы разбиваются на классы эквивалентности в соответствии с весами первого weigher, и эти классы эквивалентности сортируются в соответствии с определяющими их весами. После этого внутри каждого класса эквивалентности независимо происходит подобная сортировка элементов при помощи второго weigher и т. д. Результат такой сортировки напрямую зависит от порядка weigher-ов. При этом может оказаться, что некоторые элементы необходимо будет поднимать вверх по списку при помощи других сортировщиков. Это приводит к тому, что система теряет гибкость. Изменение же порядка weigher-ов может иметь неочевидные последствия. Исправив один сценарий, оно может сломать несколько других.

Таким образом, хотелось бы заменить стандартный сортировщик более гибким, улучшение которого не требовало бы глубокого понимания сложных взаимосвязей в программной системе.

В данной работе описан подход к улучшению сортировки в алгоритме автоматического дополнения кода с помощью алгоритмов машинного обучения. Использование таких алгоритмов позволит избежать необходимости в комбинировании эвристик вручную, оно будет происходить автоматически, необходимо лишь разработать ряд эвристик – признаков для обучения модели. Важным преимуществом данного подхода является то, что для улучшения механизма не нужно модифицировать существующий код, достаточно добавлять новые признаки и таким образом получать более качественные модели.

PSI-интерфейс структуры программы

Механизм автоматического дополнения кода в интегрированной среде разработки на платформе Intellij опирается на интерфейс структуры программы PSI. Этот интерфейс предоставляет доступ к представлению программного кода в виде абстрактного синтаксического дерева и позволяет производить операции с этим представлением, выделять составные части конструкций языка. PSI-структура представляет собой иерархию элементов, таких как psi-файл, psi-класс, psi-метод, psi-выражение и т. п. На рис. 2 показан PSI-файла для простого фрагмента кода.

С PSI-структурой взаимодействуют все основные компоненты платформы, связанные с работой с кодом, в том числе автоматическое дополнение. Реализации completion contributor-ов используют PSI-структуру для того, чтобы предложить варианты автодополнения, которые будут синтаксически корректными в данном контексте. С помощью PSI-структуры получается информация о местоположении каретки в момент вызова механизма и список элементов контекста вызова – текущей функции или класса. Некоторые варианты в списке дополнения могут иметь ссылки на psi-элементы, например элементы классов, переменных имеют отображение в psi-структуру. В то же время некоторые варианты автодополнения, например ключевые слова языка, не имеют отображения в psi-структуру.

[7] https://github.com/JetBrains/intellij-community/blob/837e67159a532cd18ec1c6418774deb8a3e24dca/platform/analysis-api/src/com/intellij/codeInsight/completion/CompletionWeigher.java

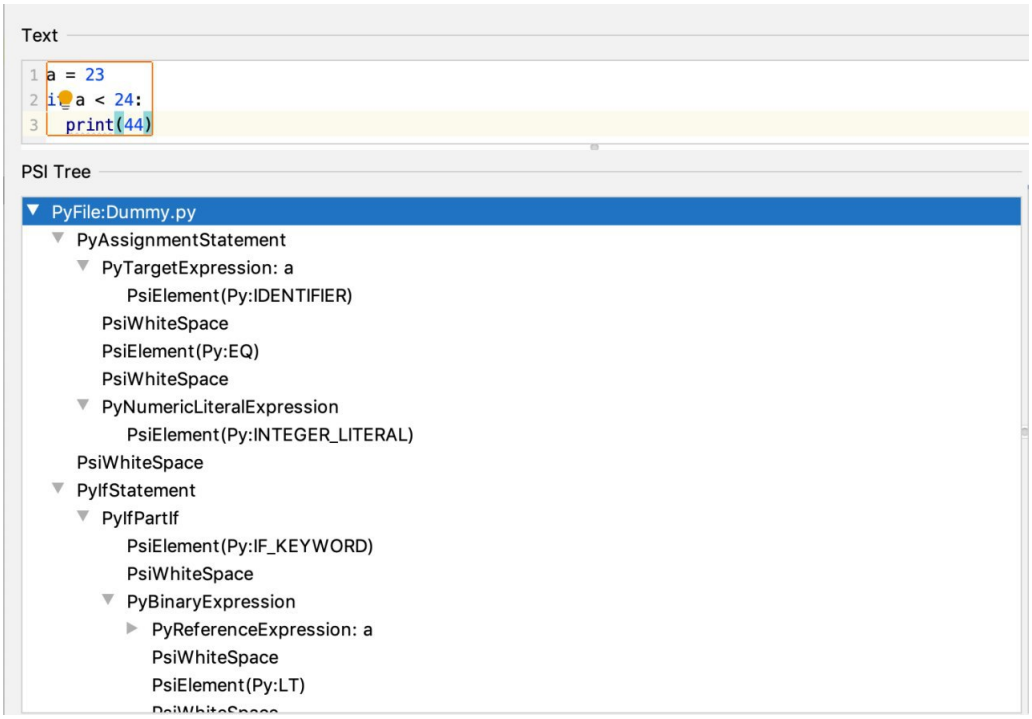

Рис. 2. Пример PSI-структуры для простого примера кода
Fig. 2. Sample PSI structure for a simple code example

Использование телеметрии

Большинство академических работ, посвященных улучшению автоматического дополнения кода, и коммерческих разработок, основаны на использовании статических данных – исходных кодов проектов, или искусственно построенных сценариях. Эти данные служат для построения некоторой модели и проверки ее качеств. С одной стороны, из таких данных действительно можно извлечь много полезной информации, с другой стороны, они не принимают во внимание поведение пользователя, не дают информации о том, где ему действительно необходима хорошая работа механизма автоматического дополнения.

Такую информацию можно получить, анализируя телеметрию – логи взаимодействия пользователя с механизмом автоматического дополнения. В них фиксируется, как именно в каждом конкретном случае был использован механизм:

- был ли он вызван явно, или же окно с вариантами появилось автоматически;
- какой префикс был введен до этого;
- отказался ли пользователь от предложенных вариантов, закрыв окно.

Понятно, что поведение пользователя в этом отношении может зависеть как от типа решаемой им задачи, так и от его индивидуальных особенностей. В идеале механизм автоматического дополнения кода должен адаптироваться к действиям и привычкам пользователя. Предлагаемый нами подход к улучшению механизма автодополнения с помощью моделей, учитывающих поведение пользователя, позволяет двигаться в этом направлении.

Полученная телеметрия может быть полезна еще и для оценки качества автодополнения. В академических работах для оценки качества чаще всего используются синтетические наборы тестов, известные специалистам. Такие наборы позволяют сравнивать работу разных систем и особенно полезны при написании обзорных статей. Но является ли такой подход полезным с точки зрения создания качественного продукта для пользователя – вопрос весьма

дискуссионный. Так, в статье Vincent J. Hellendoorn и соавторов [10] обсуждается, можно ли доверять искусственным наборам тестов и насколько сильно они могут отличаться от реальных сценариев использования механизма автодополнения.

Реализация подхода с использованием сбора логов для языка Python

Процесс поиска новых признаков для улучшения модели безграничен и зависит лишь от фантазии разработчиков. В работе предложен набор признаков, с которого можно начать оценивать работу подхода и качество модели. Выделяются две категории признаков: признаки позиции и признаки элементов. Признаки позиции описывают контекст рядом с кареткой. Этот тип признаков зависит только от положения каретки и не использует информацию о предлагаемом варианте автоматического дополнения. Признаки элемента описывают некоторые характеристики предлагаемого варианта автоматического дополнения, например общий префикс с введенной пользователем частью имени или длина предлагаемого варианта.

В текущей версии механизма для языка Python реализованы следующие признаки позиции:

- находится ли каретка в условии (рис. 3, *а*);
- находится ли каретка в управляющей части цикла (рис. 3, *б*);
- находится ли каретка после оператора if без ветки else (рис. 3, *в*);
- находится ли каретка в контексте аргументов (рис. 3, *г*);
- предыдущие ключевые слова языка в той же строке (рис. 3, *д*);
- предыдущие ключевые слова в том же столбце (рис. 3, *е*).

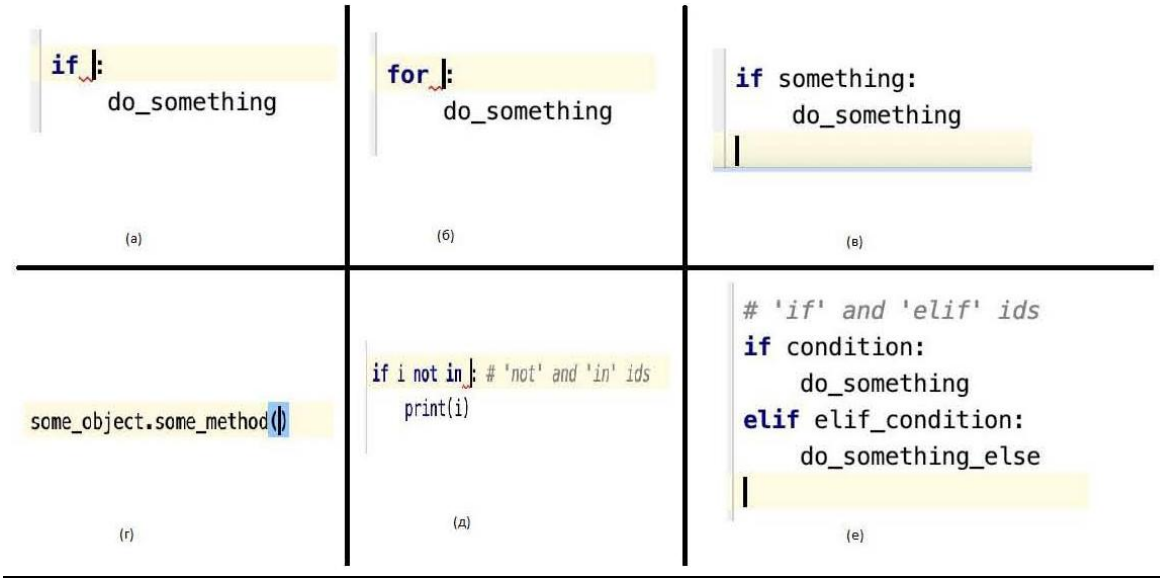

Рис. 3. Контекстные бинарные признаки

Fig. 3. Context binary signs

Для элементов сейчас реализованы следующие признаки.

- Популярность импортируемого модуля – частота его использования в кодах репозитория GitHub. Популярность основных модулей (numpy, tensorflow и т. п.) была подсчитана как количество результатов поискового запроса в GitHub по имени модуля. Значение этого признака со временем может стать неактуальным, поскольку популярности модулей меняются.

Необходимо будет обновлять информацию о популярности и обучать модель заново. Устаревания признака можно избежать, корректируя его с учетом частоты использования модуля реальными пользователями, отраженной в телеметрии.

• Популярность встроенных функций – признак, аналогичный предыдущему, но для встроенных функций Python, таких как max, min, map и др.

• Идентификатор ключевого слова: если вариант автоматического дополнения – это ключевое слово python, то значением функции будет некоторый идентификатор, назначенный этому ключевому слову (ключевые слова могут быть каким-то образом нумерованы).

• Количество элементов с таким же именем в некотором контексте, например внутри метода.

• Тип предложенного варианта, например, что он является ключевым словом, именем метода, класса или локальной переменной или других вариантов.

Некоторые признаки могут быть использованы не только для языка Python, но и для других языков, например, номер строки, номер столбца, длина совпадающего префикса, длина предлагаемого варианта, сколько раз данный вариант встречается в текущем файле и другие подобные признаки. Такие признаки должны базироваться на общих для различных языков свойствах и не должны зависеть от особенностей внутреннего представления конкретного языка. К ним также можно отнести значения, которые выдает *n*-граммная модель. Собирается информация о том, какие слова чаще всего встречаются после цепочки предыдущих слов в данном файле, и на основе этого оцениваются вероятности для элементов быть следующими в текущем контексте.

Сбор признаков и обучение модели

Значения признаков вычисляются обертками (wrappers), которые были реализованы для стандартных механизмов платформы. Эти обертки предоставляют интерфейсы для вычисления контекстных и элементных признаков для конкретных языков.

Для вычисления контекстных признаков был добавлен дополнительный completion contributor, задачей которого является не добавить новые элементы в список, а собрать вычисленные признаки со всех подключенных реализаций специальных интерфейсов для сбора таких признаков. Интерфейс предоставляет данные о позиции и контейнер для сохранения информации о контексте. Для одного языка реализаций этого интерфейса может быть несколько, и все они могут использоваться одновременно. Конкретная реализация, вычисляет и сохраняет признаки и, возможно, еще некоторую дополнительную информацию о контексте, используя информацию о позиции в PSI-структуре.

Для вычисления элементных признаков, таких как, например, длина совпадающего префикса, длина слова, добавлен специальный completion weigher. Он задает один и тот же вес каждому элементу и, таким образом, не влияет на сортировку. Помимо этого weigher-а, имеется так же интерфейс для сбора элементных признаков. Реализаций этого интерфейса может быть несколько, разделенных логически. Например, одна реализация может включать сбор признаков, основанных на имени, другая – на соответствии некоторому psi-элементенту. Помимо этого, имеется возможность использовать информацию о контексте, вычисленную при помощи реализаций сборщиков контекстных признаков. Это позволяет производить вычисление один раз, а затем использовать его результат для вычисления признаков элементов. Например, можно один раз собрать все имена, встречающиеся в функции, в которой производится автодополнение, и для каждого элемента вычислить признак – количество имен, совпавших с этим элементом. Задачей weigher является добавление признаков, собранных всеми реализациями данного интерфейса.

После того, как все признаки вычислены, они кодируются так, чтобы в них не содержалось никакой приватной информации о программе пользователя, и добавляются в логи. Признаки вычисляются в ходе сессии автоматического дополнения кода, т. е. когда пользователю

видно окно с вариантами. Если пользователь выберет некоторый элемент из списка или допечатает некоторый элемент из списка до конца вручную, то сессию можно будет использовать для обучения, построив для каждого набора вариантов элементов для дополнения, видных пользователю, целевой вектор. В этом векторе значение равно 1, если соответствующий элемент был выбран пользователем, и 0 в противном случае.

Используя такой целевой вектор и векторы признаков для каждого элемента, можно обучить модель выдавать для каждого элемента его вес – вероятность, что будет выбран именно этот элемент. Такая задача – обучение ранжированию – стандартная задача машинного обучения. Известны различные ее решения, но в нашем случае механизма автодополнения есть ограничения на размер и производительность модели: модель не должна быть слишком большой, поскольку все вычисления выполняются на клиентском персональном компьютере, и должна работать достаточно быстро, чтобы задержка появления всплывающего окна автоматического дополнения не превышала 300 миллисекунд. С учетом этих ограничений нами выбран алгоритм random forest regressor, который может быть легко адаптирован к этим требованиям. Чтобы избежать переобучения, модель обучается на наборе данных, полученных в течение одной недели, и проверяется на наборе данных, полученных в течение следующей недели.

Тестирование и результаты

Задача оценки качества механизма автоматического дополнения кода весьма нетривиальна. Можно выделить два подхода к ее решению: оценка качества на искусственных примерах и оценка качества на основе телеметрии работы механизма у реальных пользователей. Первый подход хорош своей универсальностью и возможностью сравнивать качество работы разных механизмов. Однако оценки, полученные с его помощью, не всегда адекватны, поскольку искусственные примеры могут существенно отличаться от сценариев работы реальных пользователей. У подхода на основе телеметрии такой проблемы нет, но он требует специальной поддержки со стороны системы.

Рассмотрим три способа проверки, насколько применение машинного обучения улучшает работу механизма автоматического дополнения кода:

- проверка во время обучения модели;
- проверка на искусственных примерах;
- A/B тестирование.

Проверка во время обучения модели

Пользователи были разбиты на две группы. На данных первой группы модель обучалась, а на данных второй проверялось ее качество. В качестве метрик, демонстрирующих качество модели, были выбраны метрики top-1 и top-5 – отношение количества сессий, в которых нужный элемент оказался соответственно на первом и пятом месте в списке, ко всему количеству сессий. Чем больше попаданий в эти метрики, тем выше качество. Результаты проверки представлены в табл. 1. Как можно видеть, с новым механизмом результаты улучшились для обеих метрик. Отметим, что для обучения использовались все варианты списков, видимые пользователю. Например, если он начинает вводить слово и окно с вариантами появляется только после того, как введен первый символ, это будет первый вариант списка, если после это пользователь вводит еще один символ и список вариантов фильтруется по префиксу, это уже другой вариант. Улучшение наблюдалось для всех вариантов.

Проверка на искусственных примерах

Эта проверка организована следующим образом. Выбирается проект, написанный на языке Python и запускается специальный плагин, задачей которого является открывать каждый файл с исходным кодом в проекте и для каждого слова в этом файле стирать его целиком,

вызывать автоматическое дополнение, затем скрывать его, фиксировать результат, затем печатать следующий символ стертого слова, снова фиксировать результат и т. д. Таким образом происходит имитация работы пользователя, который будто бы вызывает механизм автоматического дополнения кода для всех слов программы. Информация обо всех сессиях собирается и может быть использована как для обучения модели, так и для оценки качества работы механизма. В эксперименте для оценки использовались сессии с длинами префикса 0 и 1 (табл. 2).

Таблица 1

Сравнение топ-1 и топ-5, полученных после обучения модели

Table 1

Comparison of top-1 and top-5 metrics obtained after model training

Название метрики	Механизм работы	
	стандартный	новый
Top-1	0.547	0.735
Top-5	0.812	0.928

Таблица 2

Результат на искусственных сессиях

Table 2

Result in artificial sessions

Название метрики	Механизм работы	
	стандартный	новый
Для префикса 0		
Top-1	0.086	0.319
Top-5	0.189	0.586
Средняя позиция	97.96	36.67
Среднее время (мс)	46.9	65.6
Для префикса 1		
Top-1	0.66	0.72
Top-5	0.924	0.936
Средняя позиция	0.79	0.555
Среднее время (мс)	36.4	43.575

Результат проверки демонстрирует значительное увеличение попадания в top-1 и top-5, а также весьма существенное уменьшение средней позиции, под которой понимается усредненное по всем сессиям положение корректного варианта в списке вариантов автоматического дополнения. Чем меньше эта позиция, тем в среднем выше в списке находится необходимый элемент. Временные затраты на одну автоматическую сессию увеличились на 20 миллисекунд, что вполне допустимо. Можно заметить, что результат стандартного механизма в этой проверке гораздо хуже, чем при проверке в ходе обучения модели. Это объясняется тем, что в обучении участвуют все варианты списков, а в данном случае только варианты списков при нулевом введенном префиксе.

Таблица 2 демонстрирует улучшение работы механизма в отношении попадания в топ-1 и уменьшения средней позиции. Заметим, что и результат работы стандартного механизма на искусственных сессиях с префиксом 1 значительно лучше, чем для нулевого префикса, и даже лучше, чем усредненный результат по всем вариантам. В распределении значений при-

знака длины введенного префикса 46 % приходится на префикс 0 и 42 % – на префикс 1. Можно ожидать, что более длинные префиксы результат еще улучшат.

Проверка A/B-тестированием

A/B-тестирование – метод маркетингового исследования, направленного на выявление лучшего варианта некоторого функционала продукта. Суть метода заключается в том, что сначала выбирается некоторый элемент продукта и несколько вариантов реализации этого элемента. Создаются версии продукта с этими различными реализациями элемента и распределяются между пользователями. Данные об использовании продукта собираются, и по некоторым метрикам определяется, какая версия продукта наиболее удачна. На основе этого выбирается наилучшая по этим метрикам реализация элемента. В данной статье элемент – это алгоритм сортировки вариантов, предлагаемых во время автоматического дополнения кода, а реализациями являются стандартный механизм автоматического дополнения кода и механизм с применением машинного обучения.

A/B-тестирование точно так же, как и сбор признаков для обучения, происходит во время EAP (Early access product – версия продукта, которая предоставляется пользователям бесплатно с целью ознакомить их с новой функциональностью и отследить потенциальные проблемы, которые могут возникнуть у пользователей при добавлении этой функциональности). Половина пользователей со стандартным механизмом ранжирования вариантов автоматического дополнения кода попадают в группу A, другая половина – в группу B. На основе телеметрии, полученной от пользователей, можно сравнить две версии. В данном случае для сравнения качества используются такие метрики, как отношение долей сессий, завершившихся выбором варианта из списка, печатью варианта до конца без использования механизма автоматического дополнения (если вариант был в списке) и сессий, отмененных пользователем. Оцениваются также среднее время сессии, попадание нужного элемента в top-1 и top-5 и другие метрики.

В табл. 3 представлены результаты A/B-тестирования, полученные после включения модели. Можно заметить улучшение метрик top-1 и top-5 и средней позиции.

Таблица 3

Результаты A/B-тестирования

Table 3

Results of AB-testing

Название метрики	Механизм работы	
	стандартный	новый
Top-1	0.736	0.793
Top-5	0.943	0.967
Средняя позиция	2.8	1.85
Explicit select	0.245	0.262
Typed select	0.397	0.423
Длина сессии (мс)	1144	1057

В следующих версиях системы работа механизма будет улучшена за счет более глубокого учета контекста вызова механизма автоматического дополнения кода. Будет реализована возможность не просто оценивать качество системы в целом, а подробно изучить случаи, в которых автодополнение вызывается часто и в которых механизм требует доработки. Например, положение каретки внутри условий, циклов, в контексте аргументов функции. Для проблемных мест следует добавить новые признаки, которые улучшат работу модели.

Заключение

В статье предложен комплексный подход к решению задачи по улучшению работы механизма автоматического дополнения кода в интегрированной среде разработки PyCharm на основе машинного обучения. Описан способ сбора данных о работе механизма у реальных пользователей, построение модели с использованием алгоритмов машинного обучения и использующиеся в ней признаки. Предложенный подход реализован для языка Python. Проверка качества работы улучшенного механизма с использованием трех различных методик подтвердила обоснованность предложенного подхода. Новый механизм включен в промышленную версию среды PyCharm в качестве основного.

Гибкость реализованного механизма позволяет развивать его в различных направлениях: автоматизация сбора данных, построение моделей и оценка результатов, добавление новых признаков для обучения модели, использование более мощных библиотек для построения модели.

Список литературы / References

1. **Murphy G. C., Kersten M., Findlater L.** How are Java software developers using the Eclipse IDE? *Software, IEEE*, 2006, vol. 23 (4), p. 76–83.
2. **Hou D., Pletcher D.** Towards a Better Code Completion system by API grouping, filtering, and popularity-based ranking. In: RSSE'10: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering, 2010, p. 26–30. DOI 10.1145/1808920.1808926
3. **Han S., Wallace D. R., Miller R. C.** Code completion from abbreviated input. In: IEEE / ACM International Conference on Automated Software Engineering, 2009, p. 332–343. DOI 10.1109/ASE.2009.64
4. **Marcel Bruch**, **Martin Monperrus, Mira Mezini.** Learning from Examples to Improve Code Completion Systems. In: Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM Symposium on the Foundations of Software Engineering. Amsterdam, Netherlands, 2009, ff10.1145/1595696.1595728ff. ffhal-01575348
5. **Robbes R., Lanza M.** Improving code completion with program history. *Automated Software Engineering*, 2010, no. 17. DOI 10.1007/s10515-010-0064-x
6. **Raychev V., Vechev M., Yahav E.** Code Completion with Statistical Language Models. *ACM SIGPLAN Notices*, 2014, no. 49. DOI 10.1145/2594291.2594321
7. **Fink G**. n-Gram Models. *Automated Software Engineering*, 2014. DOI 10.1007/978-1-4471-6308-4_6
8. **Kneser R., Ney H.** Improved backing-off for m-gram language modeling. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, May 1995, vol. 1.
9. **Li Jian, Wang Yue, Lyu M., King I.** Code Completion with Neural Attention and Pointer Networks. Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18), 2018, p. 4159–4165. DOI 10.24963/ijcai.2018/578
10. **Hellendoorn V., Proksch S., Gall H., Bacchelli A.** When Code Completion Fails: A Case Study on Real-World Completions. In: ICSE 2019, p. 960–970. DOI 10.1109/ICSE.2019.00101

Материал поступил в редколлегию
Received
21.04.2020

Сведения об авторах

Матвеев Андрей Олегович, магистрант факультета информационных технологий Новосибирского государственного университета (Новосибирск, Россия)
andrey.matveev@jetbrains.com

Быстров Александр Васильевич, кандидат физико-математических наук, доцент кафедры систем информатики ФИТ НГУ (Новосибирск, Россия)
avb@iis.nsk.su

Бибаев Виталий Игоревич, программист, ООО «Интеллиджей Лабе» (Санкт-Петербург, Россия)
vitaly.bibaev@jetbrains.com

Поваров Никита Игоревич, руководитель отдела Data Science, ООО «Интеллиджей Лабе» (Санкт-Петербург, Россия)
ninkita.povarov@jetbrains.com

Information about the Authors

Andrey O. Matveev, Master's Student, Faculty of Information Technologies, Novosibirsk State University (Novosibirsk, Russian Federation)
andrey.matveev@jetbrains.com

Alexander V. Bystrov, Candidate of Physical and Mathematical Sciences Faculty of Information Technologies, Novosibirsk State University (Novosibirsk, Russian Federation)
avb@iis.nsk.su

Vitaly I. Bibaev, programmer, Intellij Labs LLC (St. Petersburg, Russian Federation)
vitaly.bibaev@jetbrains.com

Nikita I. Povarov, Data Science department head, Intellij Labs LLC (St. Petersburg, Russian Federation)
ninkita.povarov@jetbrains.com

УДК 004.9
DOI 10.25205/1818-7900-2020-18-2-76-87

Модель оценки технологических рисков предприятия

А. Н. Шендалев, О. А. Шендалева

Омский государственный университет путей сообщения
Омск, Россия

Аннотация

Рассматриваются вопросы снижения технологических рисков предприятия. Для решения этой проблемы предложена модель оценки рисков на основе анализа информации о времени и условиях эксплуатации технических средств. Предложена методика оценки критичности оборудования, и описана шкала идентификации последствий рисков для технических средств предприятия. Разработан алгоритм анализа технологических рисков и сформулированы рекомендации для дальнейшей эксплуатации объекта.

Ключевые слова

модель оценки рисков, алгоритм идентификации рисковых состояний, управление рисками

Для цитирования

Шендалев А. Н., Шендалева О. А. Модель оценки технологических рисков предприятия // Вестник НГУ. Серия: Информационные технологии. 2020. Т. 18, № 2. С. 76–87. DOI 10.25205/1818-7900-2020-18-2-76-87

Model of Technologic Risk Assessment

A. N. Shendalev, O. A. Shendaleva

Omsk State Transport University
Omsk, Russian Federation

Abstract

The aim of the work is to develop a model for describing and evaluating technological risks of an industrial enterprise, which allows you to manage risks based on their assessment and reasonably adjust the actual technical operation of the equipment. A technique for assessing the criticality of equipment is proposed and a scale for identifying the consequences of risks for the technical equipment of the enterprise is described. An algorithm for analyzing technological risks has been developed and recommendations have been formulated for the further operation of the facility.

Keywords

risk assessment model, risk identification algorithm, risk management

For citation

Shendalev A. N., Shendaleva O. A. Model of Technologic Risk Assessment. *Vestnik NSU. Series: Information Technologies*, 2020, vol. 18, no. 2, p. 76–87. (in Russ.) DOI 10.25205/1818-7900-2020-18-2-76-87

Моделирование процессов эксплуатации технических средств рассматривается в современной литературе как основной способ определения эксплуатационных параметров, предпочтительных режимов эксплуатации и технического обслуживания, включая интервалы планово-предупредительных ремонтов, а также установления параметров предельных режимов эксплуатации. На основе результатов моделирования разрабатываются прикладные алгоритмы эксплуатации технических средств, том числе при наступлении рисковых собы-

тий. Вопросам моделирования технологических процессов посвящен ряд статей [1–4], в которых разработаны математические модели высокой степени точности и детализации для различных отраслей и видов деятельности.

В данных работах анализируются не только функционирование объектов в проектных параметрах [1], но и эксплуатация технических средств при наступлении неблагоприятных внешних условий [3; 4], использовании режимов эксплуатации, не соответствующих паспортным значениям, что позволяет объективно оценивать риски наступления неблагоприятных событий. Однако данные модели основываются на значимых по объемам базах данных, обновляющихся в режиме реального времени, с использованием корпоративных баз данных. Примером реализации такой модели являются системы, использованные в ОАО «РЖД», РАО «ЕЭС России» [5; 6]. Также мониторинг открытых источников показал наличие систем управления рисками на базе имитационных моделей в ряде крупных компаний, таких как ПАО «Роснефть», «Норникель» и др., причем стоимость этих систем оценивается в десятки миллионов рублей.

Однако использование таких моделей и алгоритмов на их основе в условиях малых предприятий является экономически необоснованным в силу высоких затрат на создание, поддержание и актуализацию информационной системы. Связано это с сочетанием большого количества факторов, влияющих на проявление риска, и сравнительно малого числа устройств, подверженных этим рискам. Также малые предприятия используют преимущественно универсальное оборудование, эксплуатируемое до отказного состояния, что затрудняет выбор значений нормального режима эксплуатации и моментов наступления риска. Наконец, проявление технологических рисков для крупной компании не означает остановку деятельности или наступление финансовых кризисов, в то время как для малых предприятий технологический риск неразрывно связан с кризисным состоянием.

В связи с вышесказанным разработка модели и алгоритма описания и оценки технологических рисков, адаптированных для возможностей малых предприятий, является актуальной задачей. Новизна предложенного алгоритма заключается в одновременном использовании экспертных оценок и количественных значений, учитывающих сроки и интенсивность эксплуатации технических средств, не требующих высокозатратной системы учета и анализа.

Управление технологическими процессами предприятия в настоящее время существенно ограничено экономическим подходом, согласно которому внепроизводственные затраты (т. е. затраты, не имеющие непосредственного отношения к производству и реализации продукции) должны быть сведены к минимуму [7; 8]. Реализация такого требования фактически приводит к ситуации, когда процессы, значимые для технического состояния производства, такие как планово-предупредительное обслуживание оборудования, узлов и агрегатов, регламентная замена старогодных узлов и деталей технических средств и подобные им, оказываются экономически нецелесообразными и исключаются из перечня производственных процессов. В частности, в ОАО «РЖД», а также ряде вертикально интегрированных компаний уже более 10 лет идет эксплуатация подвижного состава по фактическому состоянию объектов, а на малых и средних промышленных предприятиях распространена практика эксплуатации оборудования, машин, транспортных средств и прочих производственных объектов до отказного состояния [9; 10].

С одной стороны, подобный подход обеспечивает максимальное использование ресурса оборудования и объектов инфраструктуры, а также потенциальное увеличение межремонтных периодов. С другой стороны, наряду с положительными эффектами имеется еще и ряд потенциально отрицательных вероятностных эффектов:

- с течением времени повышается вероятность выхода оборудования из строя непосредственно в ходе эксплуатации и, как следствие, остановки технологического процесса или отдельной его фазы;
- имеется вероятность срыва точности поставок и связанная с ней необходимость увеличения межоперационных запасов;

- вероятность порчи материалов и комплектующих, находящихся на обработке;
- вероятное снижение эксплуатационных характеристик оборудования, таких как точность, мощность и др.

Иначе говоря, практика подобной эксплуатации обладает высокой степенью технологических рисков.

Модель описания и оценки рисков промышленных предприятий должна, по нашему мнению, включать в себя два укрупненных блока, реализующих основную идею модели. Первый блок связан с идентификацией рисковых технических средств, т. е. дифференциация объектов на группы по уровням риска. Второй блок отвечает за формирование актуальных оценок рисков, их анализ и систематическое проведение пересмотра рисков и выработку рекомендаций для дальнейшей эксплуатации технических средств предприятия.

Функциональная модель описания и оценки рисков приведена на рис. 1. Блок идентификации риска включает в себя:

- идентификацию рисковых событий;
- идентификацию параметров событий;
- получение достоверной исходной информации;
- проведение необходимого анализа имеющейся информации;
- формирование исходных данных для дальнейшего выбора оптимальных решений по обработке риска.

Блок анализа и оценки риска, в свою очередь, состоит из следующих шагов:

- определение области применения технического средства, сбор и анализ информации об условиях эксплуатации объекта, таких как природные условия эксплуатации, предусмотренные технологическим процессом время межпартионных ожиданий и среднестатистическое время простоя;
- идентификация рисков по группам;
- оценка величины риска;
- оценивание риска;
- принятие обоснованных решений при оценивании риска.

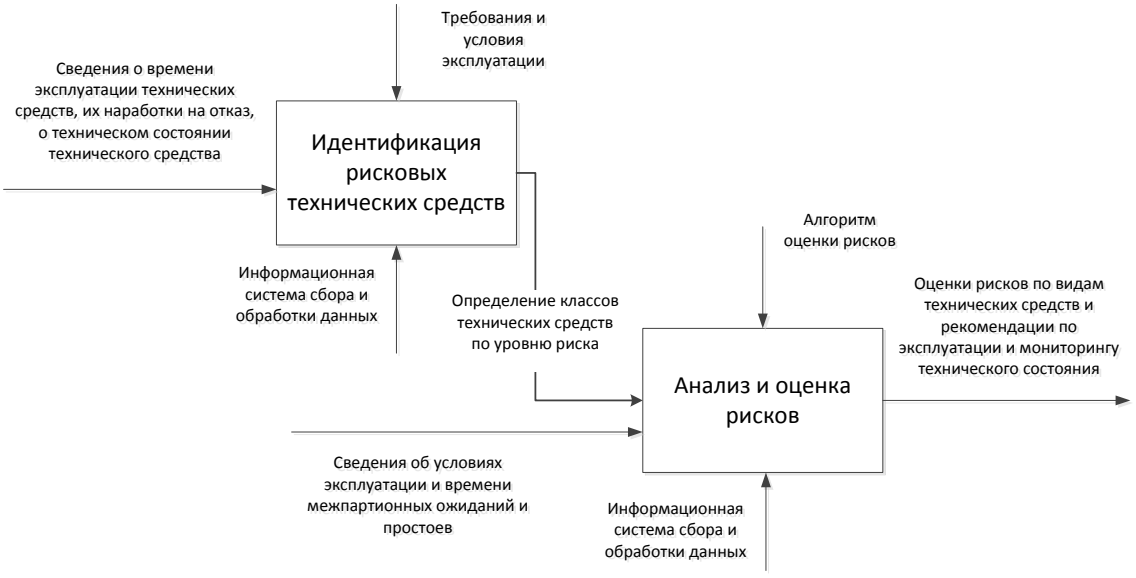

Рис. 1. Функциональная модель оценки рисков
Fig. 1. Functional Risk Assessment Model

Как видно из приведенного рисунка, в основе модели лежит база данных используемых технических средств, сведенная в единую информационную систему, и содержащая обобщенную информацию об условиях и интенсивности эксплуатации технических средств. Данные этой базы должны поддерживаться всегда в актуальном состоянии и перманентно обновляться. На основе оценок технического состояния объекта проводится его идентификация, т. е. определяется его принадлежность к одному из классов рисковости, а затем по заданному алгоритму оцениваются все риски.

Результатом реализации и использования модели оценки рисков является:

- идентификация рискового средства на основе сравнения условий эксплуатации объекта с соответствующими объекту требованиями безопасности;
- уточнение информации об основных опасностях;
- разработка рекомендаций по обоснованию или изменению нормативных требований, по вопросам лицензирования, определения частоты проверок состояния объекта и его безопасности;
- совершенствование руководств по эксплуатации и техническому обслуживанию, планов локализации опасностей;
- оценка эффекта изменений в организационной структуре, способах практической работы и технического обслуживания в отношении показателей безопасности.

Технологический риск как категория оценки технического состояния имеет ряд существенных особенностей, которые необходимо учесть при разработке модели оценки риска [11].

1. Риск является прямым следствием эксплуатации, а также неблагоприятных событий, возникающих в процессе эксплуатации технического средства на промышленном предприятии. Как следствие, превышение сроков бесперебойной работы, а также воздействие на риск таких неблагоприятных факторов, как отклонение режимов эксплуатации, использование материалов и комплектующих, не соответствующих технологическим требованиям, минимизация технологических перерывов, приводит к увеличению риска.

2. Реализация алгоритма оценки риска не предполагает одномоментную оценку. Алгоритм носит характер циклического повторения, в котором рассматриваются отдельные технические средства. Полученные результаты меняются во времени.

3. При анализе рисков не рассматривается взаимное влияние технических средств. Воздействие рассматривается как статистический параметр и учитывается при анализе рисков в виде соответствующего коэффициента.

4. Данный алгоритм не распространяется на стохастические риски, такие как форс-мажор, а также на риски экономической природы.

Алгоритм идентификации рисковых технических средств приведен на рис. 2.

Набор исходных данных для реализации модели формируется по видам технических средств и представляет собой систематически обновляемую базу данных о режимах эксплуатации оборудования. Вид базы данных приведен в табл. 1.

Нормативное время эксплуатации нового оборудования определяется как нормативно установленное значение срока службы технического средства (по видам технических средств). Нормативное время эксплуатации для объектов, чей срок службы превысил свою норму, можно определить исходя их имеющихся статистических данных о времени работы до отказа.

$$t_{\text{проект}} = \frac{t_{\text{отказ}}}{k}, \tag{1}$$

где $t_{\text{проект}}$ – нормативное время эксплуатации до условного предельного состояния технического средства; $t_{\text{отказ}}$ – среднее время межотказного периода; k – коэффициент запаса.

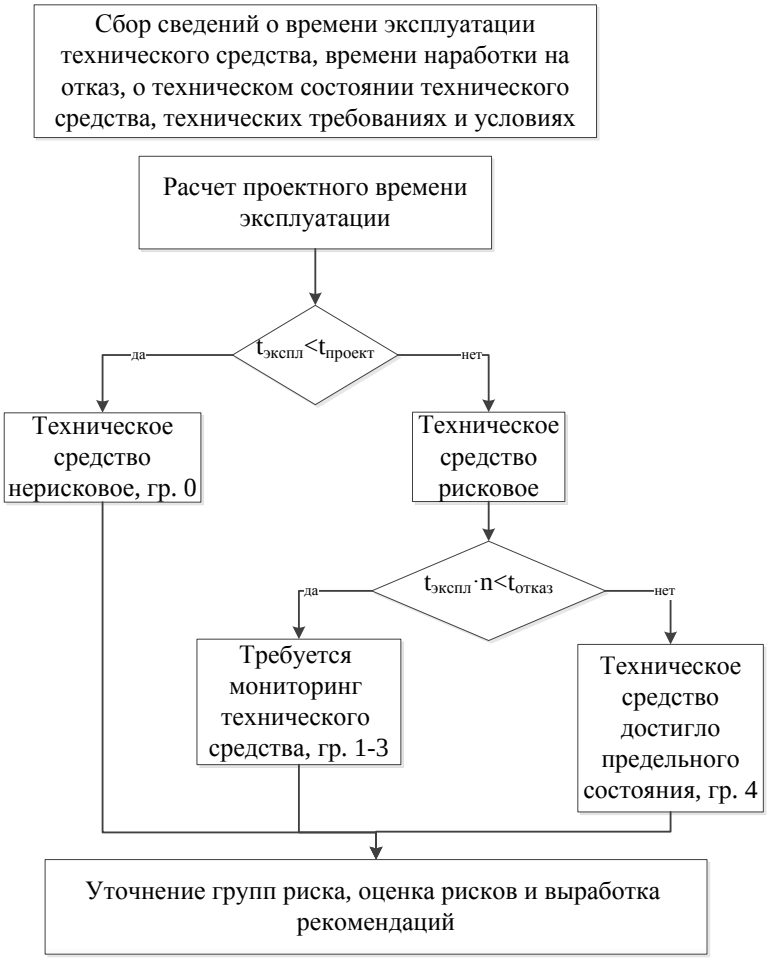

Рис. 2. Алгоритм идентификации рисковых технических средств
Fig. 2. Algorithm for identification of risky technical means

Таблица 1

Структура базы данных по сведениям о технических средствах

Table 1

Structure of database for information about technical means

Наименование технического средства	Нормативный срок службы	Фактический срок службы	Продолжительность эксплуатации до текущего нормативного ремонта				Запас прочности
			Вид ремонта 1		Вид ремонта 2		
			норма	факт	норма	факт	

Коэффициент запаса k рекомендуется принимать в диапазоне [1,1; 1,3]. Значение коэффициента определяется опытным путем и зависит от количества внеплановых ремонтов, проведенных для данного типа оборудования. В случаях если для оборудования проводились только плановые ремонты, рекомендуется принимать $k = 1{,}1$. Для технических средств, которые ремонтировались неоднократно, а стоимость внеплановых ремонтов была значительной, значение коэффициента запаса будет максимальным.

Время межотказного периода фиксируется по типам отказов и характеризует время прохождения между двумя типами отказов. В случае если техническое средство новое и ранее отказов не наблюдалось, значение k следует принять как фактическое время эксплуатации. Также нужно учитывать, что под отказами понимается событие, приведшее к остановке технологического процесса либо существенному падению мощности.

Для идентификации технических средств, которые могут достичь условного предельного состояния следует воспользоваться формулой

$$P = \frac{t_{\text{экспл}}}{t_{\text{проект}} \cdot n}, \tag{2}$$

где P – вероятность достижения условного предельного состояния; $t_{\text{экспл}}$ – фактическое время эксплуатации; $t_{\text{проект}}$ – нормативное время эксплуатации; n – запас прочности.

Значение запаса прочности определяется либо на базе нормативной документации (ГОСТ 21354-87, РД 10-249-98 и др.), либо на базе опытно-статистических данных. Условно для промышленных предприятий значение n можно принять в интервале 2–10. Значение запаса прочности зависит от множества факторов, а именно от надежности, заложенной при его проектировании, условий эксплуатации, от настоящего технического состояния и др.

Некоторою сложность для расчета будут представлять технические средства, исчерпавшие свой нормативный срок эксплуатации. Проблема заключается в том, что, руководствуясь приведенными формулами, данные технические средства будут гарантированно рассматриваться как рискованные. Авторы предлагают в данном случае руководствоваться подходом, когда для технических средств находящихся в удовлетворительном техническом состоянии нормативный срок эксплуатации определяется на базе остаточного ресурса. Значение вероятности отказа для подобных технических средств следует представить в виде

$$P = \frac{t_{\text{экспл}}}{t_{\text{проект}} \cdot n} \cdot \frac{t_{\text{отказ.ср}}}{t_{\text{отказ.факт}}}, \tag{3}$$

где $t_{\text{отказ.факт}}$ – фактическое время отказа; $t_{\text{отказ.ср}}$ – среднее время отказа по данному типу технических средств и виду отказа.

Полученное значение вероятности отказа необходимо сравнить с критическим уровнем, значение которого устанавливается руководством предприятия. Проведя мониторинг [12–15], авторы делают вывод о том, что критическое значение должно находиться в интервале 3–5 %.

Предлагаемый алгоритм модели идентификации рисков позволяет отсеять нерискованные технические средства, а также разделять в зависимости от эксплуатационного состояния технических средств на объекты рискованные и нерискованные (нуждающиеся в мониторинге, либо допущенные к эксплуатации и / или хранению без ограничений).

Второй блок модели оценки риска предполагает анализ рисковых технических средств на предмет разработки корректирующих и предупреждающих мероприятий. Зная перечень рискованного оборудования со стратификацией по уровню риска и типу технических средств, можно определить критичность данных рисков. Необходимость данного действия вызвана потребностью формирования плана реагирования на возможные риски.

Для оценки критичности рискованного оборудования могут быть использованы методы:

- оценка частоты возникновения события в прошлом на основе статистических данных (данные, накопленные за некоторый период эксплуатации рассматриваемого объекта инфра-

структуры, статистические данные о происшествиях и других событиях и т. п.) и прогнозирование частоты, с которой это событие может возникать в будущем;

- оценка критичности рисков на основе данных об отказах технических средств, произошедших за определенный период времени, приходящихся на единицу продукции / услуги;
- прогнозирование критичности событий с использованием анализа диаграммы возможных отказов объекта инфраструктуры (анализ «дерева отказов») и анализа диаграммы возможных последствий данного отказа («дерева событий»);
- оценка на основе мнения экспертов.

При проведении экспертных оценок следует учитывать любую доступную информацию об объекте инфраструктуры. Целью анализа является оценка убытков, возникающих при наступлении рисков по техническим средствам. Для оценки убытков предлагается использовать следующую шкалу (табл. 2).

Таблица 2

Шкала идентификации последствий рисков технических средств

Table 2

Scale of identification of the consequences of risks of technical means

Уровень тяжести последствий	Значение коэффициента тяжести	Последствия по видам риска
Катастрофический	20–16	Гибель одного или более человек или тяжкие телесные повреждения 5 или более человек, связанных с функционированием технического средства предприятия; оборудование повреждено до степени исключения из инвентарного парка; нанесен критический ущерб объекту инфраструктуры
Критический	15–11	Тяжкие телесные повреждения до 5 человек, связанные с функционированием технического средства предприятия; повреждение оборудование, требующее проведение капитального ремонта для восстановления его работоспособности; нанесен ущерб объекту инфраструктуры существенного значения; потеря партии продукции, срыв поставки продукции
Несущественный	10–6	Вред средней тяжести, нанесенный здоровью сотрудника; повреждение оборудования предприятия, требующее проведения регламентного ремонта для восстановления его работоспособности; нанесен ущерб объекту ведомого значения; срыв сроков поставок
Незначительный	5–1	Легкий вред здоровью сотрудника; повреждение оборудования предприятия, требующее проведение текущего ремонта для восстановления его работоспособности; нанесен незначительный ущерб объекту инфраструктуры

Тяжесть повреждений регламентируется нормативно-правовыми актами РФ. Значение каждой категории риска устанавливается руководством организации на основании экспертных оценок применительно к собственному предприятию. Например, для методик анализа рисков, применяемых на железной дороге, в качестве критичного значения принята величина в 5 000 МРОТ, для незначительного риска – менее 500 МРОТ.

Ориентировочные значения для данной шкалы могут быть определены на основании экстраполяционного анализа статистических данных. Так, используя статистические данные за 2019 год и экстраполируя приведенные для РЖД значения, можно предположить, что критическим значением ущерба для малого предприятия будет значение от 200 МРОТ, а незначительным – менее 20 МРОТ.

Основной задачей анализа технических средств выступает оценка критичности рискового оборудования и технических устройств. Оценка критичности U ведется по формуле

$$U = \frac{t_{\text{экспл}} \cdot K_u \cdot Z}{t_{\text{проект}} \cdot n}, \tag{4}$$

где K_u – коэффициент учета тяжести последствий отказа технического средства; Z – суммарные затраты на создание технического средства, включая внереализационные затраты, а также затраты на подготовительно-заключительные работы.

Значение K_u может находиться в интервале от 1 до 20 и назначается исходя из критичности событий (см. табл. 1).

Алгоритм анализа второго блока и разработки рекомендаций отражен на рис. 3.

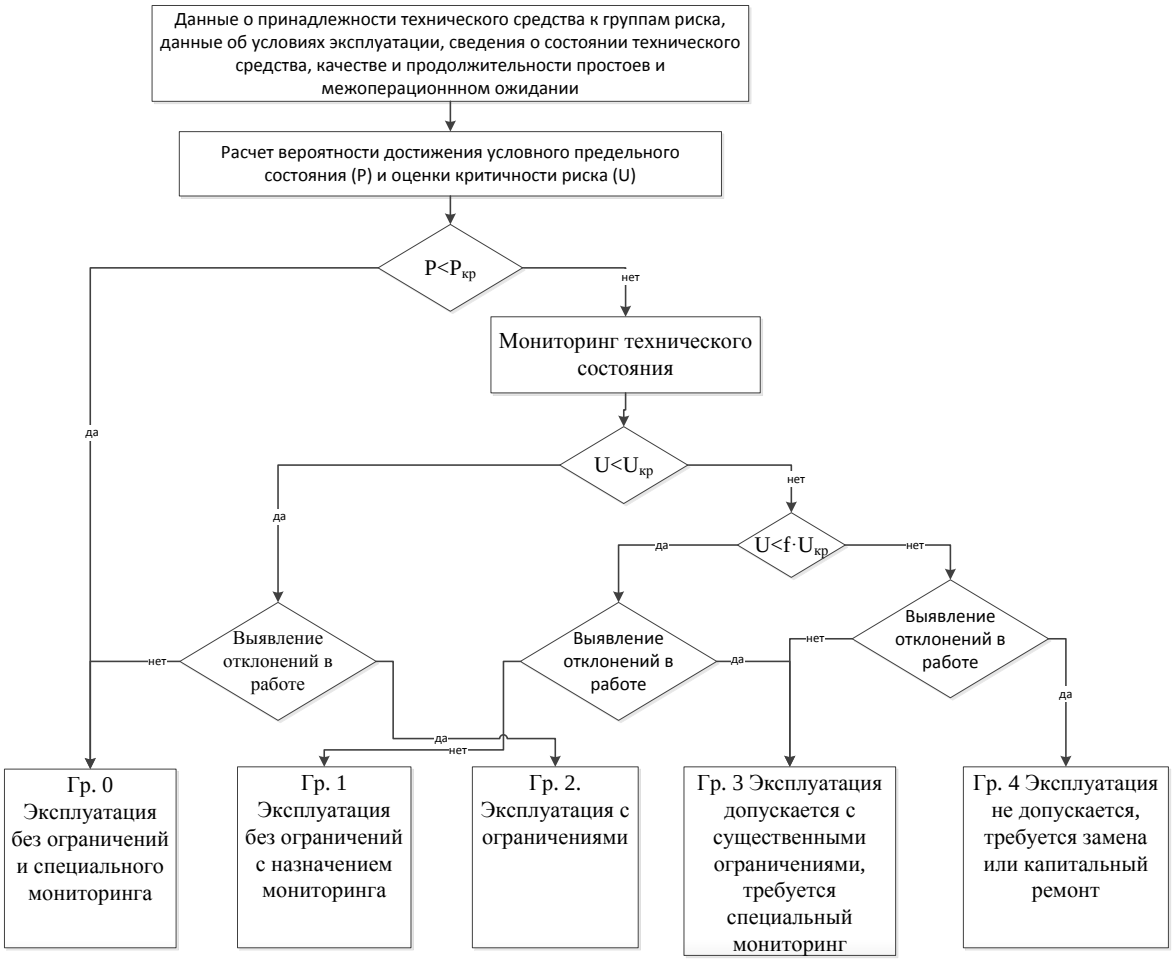

Рис. 3. Алгоритм анализа риска и выработки рекомендаций

Fig. 3. Algorithm for risk analysis and development of recommendations

Реализация данного алгоритма позволяет сформировать укрупненные группы оборудования и разработать корректирующие и предупреждающие мероприятия для группы. Каждая группа представляет собой качественно выраженное состояние технического средства, характеризующее возможность его дальнейшей эксплуатации. Важно отметить, что данные группы связаны между собой, и реализация модели идентификации и оценки рисков позволяет отразить переход технических средств из одной группы в другую, что создает наглядную картину динамики риска.

Таким образом, предприятие имеет возможность разработки комплекса мероприятий по своевременному реагированию на изменение состояния объектов. Меры по минимизации ущербов на предприятии следует подразделять на действия, направленные на уменьшение экономических, социальных и экологических ущербов. Снижение экономических ущербов предполагает усилия по защите основных фондов от действия поражающих факторов возможных рисковых ситуаций, повышению устойчивости технологических процессов к отказам. Снижение социальных ущербов обеспечивается за счет защиты персонала организации, а также населения прилегающих территорий, рационального размещения объектов, создание санитарных зон вокруг критических технических средств. Снижение экологических ущербов включает усилия по защите водоемов, атмосферы, почвенного покрова от попадания опасных веществ и энергий, выделяющихся в ходе эскалации возможных рисков.

В основе практических мер по снижению ущерба от потенциальных аварий лежат конкретные превентивные мероприятия научного, инженерно-технического и технологического характера**,** осуществляемые для парирования природных и техногенных опасностей [16; 17]**.** Значительная часть этих мероприятий проводится в рамках инженерной, радиационной, химической и медицинской защиты населения и территорий, прилегающих к площадкам размещения объекта.

Мероприятия по снижению ущербов определяются спецификой технического средства. Однако указанные мероприятия имеют общие научные, инженерно-конструкторские, технологические основы, служащие методической базой для снижения ущербов. В качестве примеров таких мер, могут быть названы:

- совершенствование технологических процессов;
- повышение надежности технологического оборудования и эксплуатационной надежности в целом;
- своевременное обновление основных фондов;
- использование качественной конструкторской и технологической документации;
- применение высококачественного сырья, материалов, комплектующих изделий;
- привлечение высококвалифицированного персонала;
- создание и использование эффективных систем технологического контроля и технической диагностики, безаварийной остановки производства, локализации аварийных ситуаций и многое другое.

Предлагаемая модель предполагает систематическое повторение процессов идентификации рисков, последующего анализа полученных данных и разработки корректирующих и предупреждающих мероприятий. Более того, значения вероятности наступления событий, параметры надежности и другие элементы информационной модели идентификации и оценки риска за прошедшие периоды могут быть использованы в качестве базовых значений для периодов последующих. Использование модели описания и оценки технологических рисков промышленного предприятия позволяет управлять рисками на основе их оценки и обоснованно регулировать фактический срок эксплуатации технических средств. Кроме того, использование предложенной информационной модели позволяет установить приоритеты в организации ремонта технических средств, в том числе и оборудования, относящегося к группе старогодных технических средств.

Список литературы

1. **Детина Е. П., Ермак И. С.** Моделирование системного подхода к управлению технологическими и эксплуатационными рисками на объектах производства и распределения газа // Естественные и технические науки. 2019. № 9. С. 152–157.
2. **Голева А. И., Стороженко Н. Р., Потапов В. И., Шафеева О. П.** Математическое моделирование отказоустойчивости информационных систем // Вестник НГУ. Серия: Информационные технологии. 2019. Т. 17, № 4. С. 33–45.
3. **Журавлев С. С., Рудометов С. В., Окольнишников В. В., Шакиров С. Р.** Применение модельно-ориентированного проектирования к созданию АСУ ТП опасных промышленных объектов // Вестник НГУ. Серия: Информационные технологии. 2018. Т. 16, № 4. С. 41–55.
4. **Степин Ю. П., Бледных Е. Н.** Системное моделирование, оптимизация, оценка и анализ рисков и эффективности функционирования нефтегазовых производственных систем // Автоматизация, телемеханизация и связь в нефтяной промышленности. 2020. № 4. С. 26–34.
5. **Воконян Е.** Индустриальный интернет: настоящее или будущее отечественной энергетики? // Энергетика и промышленность России. 2016. № 1. С. 24–26.
6. **Гапанович В. А., Шубинский И. Б., Проневич О. Б.** Система управления рисками крупных компаний. Практика оценки рисков в ОАО РЖД и направление развития // Проблемы анализа риска. 2018. № 2 (15).
7. **Рыжкова Е. В., Иода Е. В.** Особенности управления рисками промышленного предприятия // Социально-экономические явления и процессы. 2015. № 9. С. 146–152.
8. **Freudenburg W. R.** Perceived risk, real risk: social science and the art of probabilistic risk assessment. *Science*, 1988, no. 4875, p. 44–49.
9. **Хазиева А. Т., Хайруллина А. Д.** Управление производственными рисками промышленного предприятия // Электронный научный журнал. 2016. № 10-3 (13). С. 188–192.
10. **Гилязутдинова И. В., Поникарова А. С.** Управление промышленными рисками инновационной деятельности на предприятиях нефтехимической промышленности // Вестник Казанского технологического университета. 2009. № 2. С. 368–375.
11. **Николаева А. А.** Систематизация рисков промышленных предприятий // Economic Theory. 2015. № 5 (126). С. 76–80.
12. **Asimit A. V., Badescu A. M., Verdonck T.** Optimal risk transfer under quantile-based risk measurers. *Insurance: mathematics and economics*, 2013, no. 1, p. 252–265.
13. **Кунин В. А.** Управление рисками промышленного предпринимательства (теория, методология, практика). СПб.: СПбАУЭ, 2011. 144 с.
14. **Махметова А. Е., Киселева М. В.** Оценка рисков в управлении качеством продукции на промышленных предприятиях // Экономика в промышленности. 2017. № 2. С. 147–152.
15. **Ларинина Т. И.** Методические подходы к управлению инновационной инфраструктурой промышленного предприятия на основе управления рисками // Nauka-Rastudent.ru. 2016. № 12. С. 3–12.
16. **Диасамидзе М. А., Хакимова Г. Р.** Качественные методы оценки рисков в системе риск-менеджмента промышленных предприятий // Международный технико-экономический журнал. 2017. № 1. С. 13–18.
17. **Бадалова А. Г., Пановский В. Н.** Управление рисками при реализации проектов технического перевооружения промышленных предприятий // Вестник МГТУ. 2015. № 1 (32). С. 117–124.

References

1. **Detina E. P., Ermak I. S.** Modeling a systematic approach to managing technological and operational risks at gas production and distribution facilities. *Natural and Technical Sciences*, 2019, no. 9, p. 152–157. (in Russ.)
2. **Goleva A. I., Storozhenko N. R., Potapov V. I., Shafeeva O. P.** Mathematical modeling of fault tolerance of information systems. *Vestnik NSU. Series: Information Technology*, 2019, vol. 17, no. 4, p. 33–45. (in Russ.)
3. **Zhuravlev S. S., Rudometov S. V., Okolishnikov V. V., Shakirov S. R.** Application of model-oriented design to the creation of industrial control systems for hazardous industrial facilities. *Vestnik NSU. Series: Information Technology*, 2018, vol. 16, no. 4, p. 41–55. (in Russ.)
4. **Stepin Yu. P., Blednykh E. N.** System modeling, optimization, risk assessment and analysis of the functioning of oil and gas production systems. *Automation, telemechanization and communication in the oil industry*, 2020, no. 4, p. 26–34. (in Russ.)
5. **Vokonyan E.** Industrial Internet: the present or future of domestic energy? *Energy and Industry of Russia*, 2016, no. 1, p. 24–26. (in Russ.)
6. **Gapanovich V. A., Shubinsky I. B., Pronevich O. B.** The risk management system of large companies. Risk assessment practice in Russian Railways and development direction. *Problems of risk analysis*, 2018, no. 2 (15). (in Russ.)
7. **Ryzhkova E. V., Ioda E. V.** Features of risk management of an industrial enterprise. *Socio-economic phenomena and processes*, 2015, no. 9, p. 146–152. (in Russ.)
8. **Freudenburg W. R.** Perceived risk, real risk: social science and the art of probabilistic risk assessment. *Science*, 1988, no. 4875, p. 44–49.
9. **Khaziev A. T., Khayrullina A. D.** Industrial risk management of an industrial enterprise. *Electronic scientific journal*, 2016, no. 10-3 (13), p. 188–192. (in Russ.)
10. **Gilyazutdinova I. V., Ponikarova A. S.** Management of industrial risks of innovative activity at the enterprises of the petrochemical industry. *Bulletin of Kazan Technological University*, 2009, no. 2, p. 368–375. (in Russ.)
11. **Nikolaev A. A.** Systematization of risks of industrial enterprises. *Economic Theory*, 2015, no. 5 (126), p. 76–80. (in Russ.)
12. **Asimit A. V., Badescu A. M., Verdonck T.** Optimal risk transfer under quantile-based risk measurers. *Insurance: mathematics and economics*, 2013, no. 1, p. 252–265.
13. **Kunin V. A.** Risk management of industrial entrepreneurship (theory, methodology, practice). St. Petersburg, SPbAUE Press, 2011, 144 p. (in Russ.)
14. **Makhmetova A. E., Kiseleva M. V.** Risk assessment in product quality management at industrial enterprises. *Economics in Industry*, 2017, no. 2, p. 147–152. (in Russ.)
15. **Larinina T. I.** Methodological approaches to managing the innovation infrastructure of an industrial enterprise based on risk management. *Nauka-Rastudent.ru*, 2016, no. 12, p. 3–12. (in Russ.)
16. **Diasamidze M. A., Khakimova G. R.** Qualitative methods for assessing risks in the risk management system of industrial enterprises. *International Technical and Economic Journal*, 2017, no. 1, p. 13–18. (in Russ.)
17. **Badalova A. G., Panovsky V. N.** Risk management during the implementation of projects for the technical re-equipment of industrial enterprises. *Vestnik MSTU*, 2015, no. 1 (32), p. 117–124. (in Russ.)

Материал поступил в редколлегию
Received
17.04.2020

Сведения об авторах

Шендалев Александр Николаевич, канд. экон. наук, доцент кафедры «Экономика транспорта, логистика и управление качеством» Омского государственного университета путей сообщения ОмГУПС (Омск, Россия)
shendalev@mail.ru

Шендалева Ольга Анатольевна, канд. техн. наук., доцент кафедры «Информатика и компьютерная графика» Омского государственного университета путейсообщения ОмГУПС (Омск, Россия)
oa_shendaleva@mail.ru

Information about the Authors

Aleksandr N. Shendalev, Cand. Econ. associate Professor, Department of transport Economics, logistics and quality management, Omsk State University of Railway Engineering (Omsk, Russian Federation)
shendalev@mail.ru

Olga A. Shendaleva, candidate of technical Sciences associate Professor of the Department of Informatics and computer graphics, Omsk State University of Railway Engineering (Omsk, Russian Federation)
oa_shendaleva@mail.ru

Правила оформления текста рукописи

Авторы представляют статьи на русском или английском языке объемом от 0,5 авторского листа (20 тыс. знаков) до 1 авторского листа (40 тыс. знаков), включая иллюстрации (1 иллюстрация форматом 190 × 270 мм = 1/6 авторского листа, или 6,7 тыс. знаков). Публикации, превышающие указанный объем, допускаются к рассмотрению только после индивидуального согласования с редакцией журнала.

Текст рукописи должен быть представлен в редколлегию в виде файла MS Word (.doc, .docx). Гарнитура Times New Roman, размер шрифта 11, межстрочный интервал 1, размеры полей – стандартные значения текстового редактора. Форматирование – выравнивание по ширине страницы, переносы слов включены, каждый новый абзац начинается с красной строки. Не допускается ручное форматирование абзацев (пробелами, лишними переводами строк, разрывами страниц).

Структура статьи

- Индекс УДК (универсальной десятичной классификации). Выравнивание по левому краю
- Название статьи. Выравнивание по центру, полужирный шрифт
- ФИО авторов (инициалы, фамилия). Выравнивание по центру, полужирный шрифт
- Места работы всех авторов. Выравнивание по центру, курсив
- Аннотация статьи
- Ключевые слова, не более 10
- Благодарности, сведения о финансовой поддержке
- Название статьи **на английском языке**. Выравнивание по центру, полужирный шрифт
- ФИО авторов **на английском языке** (инициалы, фамилия). Выравнивание по центру, полужирный шрифт
- Места работы авторов **на английском языке**. Выравнивание по центру, курсив
- Аннотация статьи **на английском языке (Abstract),** 200–250 слов
- Ключевые слова **на английском языке (Keywords)**, не более 10
- Благодарности, сведения о финансовой поддержке **на английском языке**, если есть соответствующий раздел на русском языке **(Acknowledgements)**
- Основной текст
- Список литературы / **References**
- Сведения об авторах

Требования к оформлению основного текста и иллюстративных материалов

Основной текст должен быть представлен в структурированном виде, рекомендуется использовать подзаголовки – например: Введение, Методика..., Выводы, Результаты, Заключение.

Подзаголовки отделяются и набираются полужирным шрифтом. В целях выделения частей текста и отдельных слов и словосочетаний допускается использование курсива или полужирного шрифта. Подчеркивание, разрядка, изменение основного кегля и выделение цветом не используются.

Иллюстрации к рукописи статьи должны быть приложены в виде отдельных файлов. При этом в тексте должно содержаться включенное изображение с указанием имени файла. Все

иллюстрации, содержащие схемы, графики, алгоритмы и т. п., должны быть представлены в векторном виде (.ai, .eps, .cdr). Скриншоты и другие растровые изображения должны быть представлены в максимально высоком качестве, без каких-либо потерь и искажений (.jpg, .tif). Все иллюстрации должны иметь подрисуночную подпись – свое название. Надписи к таблицам и подписи к иллюстрациям приводятся **на двух языках (русском и английском).**

Примеры:

Рис. 1. Диаграмма производительности...
Fig. 1. Performance diagram...

Таблица 1

Сравнение алгоритмов...

Table 1

Comparison of algorithms...

Нумерация последовательная и неразрывная от начала статьи. Не допускается использование других наименований, кроме «Рис.» / «Fig.», «Таблица» / «Table», и усложнение нумерации (например, «Рис. 3.2.»). Ссылка на иллюстрацию в тексте должна быть приведена в круглых скобках, например: (рис. 1), (табл. 1).

Формулы должны быть набраны с использованием редактора MathType либо встроенного редактора формул MS Word. Кегль основных символов – 11, греческие символы набираются прямым шрифтом, латинские – курсивом. Нумеруются только те формулы, на которые автор ссылается в тексте.

Abstract

Аннотация статьи на английском языке (Abstract) не должна быть дословным переводом русскоязычной аннотации. Раздел Abstract, как и основной текст, должен быть структурирован, в нем должно содержаться описание цели работы, методов исследования, научной значимости, выводов / результатов. Требуется качественный перевод на английский язык (при необходимости просим авторов обращаться к профессиональным переводчикам). **Объем Abstract 200–250 слов.**

Список литературы / References

Список литературы и список литературы на английском языке (References) размещаются в общем разделе. Рекомендуемое количество цитируемых в статье источников – не менее 10, в список желательно включать ссылки на актуальные работы по теме исследования, особенно в иностранных периодических изданиях.

В тексте статьи ссылки на литературу указываются цифрами в квадратных скобках, при необходимости указываются номера страниц, например: [2; 3. С. 15].

Список литературы нумеруется в порядке цитирования и оформляется в соответствии с ГОСТ Р 7.0.5-2008 на библиографическое описание (знаки тире в описании опускаются). Ссылки на неопубликованные работы, а также на Интернет-ресурсы (кроме электронных изданий, поддающихся библиографическому описанию) оформляются в виде сноски.

В Список литературы ссылки на источники следует включать на оригинальном языке опубликования. Каждый источник должен быть также оформлен на английском языке (References) по международному стандарту для публикаций в области информатики IEEE Style со следующими отличиями:

- инициалы авторов указываются после фамилии;
- название статьи не берется в кавычки, отделяется точкой;
- отсутствует союз «and» перед фамилией последнего автора;
- в диапазоне страниц указывается одна «p» (вместо «pp. 2–9» – «p. 2–9»);
- год издания указывается после места издания (для книг) и сразу после названия журнала (для периодики).

Перевод источника на английский язык:

- если источник имеет выходные данные на английском языке, то для формирования References **следует использовать именно эти данные**;
- если оригинальная публикация не содержит выходных данных на английском языке, то допускается транслитерация названия материала на латинский алфавит в сочетании с переводом на английский язык в квадратных скобках. В конце описания указывается, на каком языке написана эта работа, например, (in Russ.). При транслитерации можно воспользоваться Интернет-ресурсом http://ru.translit.ru/, рекомендуется выбрать стандарт BSI. Место издания не транслитерируется, указывается полностью на английском языке, например: Moscow. Название издательства / издателя, как правило, транслитерируется. Для журналов, у которых есть официальное название на английском языке, – использовать его (проверить на сайте журнала, или, например, в библиотеке WorldCat), если названия на английском языке нет, использовать транслитерацию по системе BSI. Не следует самостоятельно переводить названия журналов.

Если у цитируемого источника есть **цифровой идентификатор DOI** (https://search.crossref.org), его требуется обязательно указывать в конце библиографической ссылки.

Примеры оформления ссылок. Каждый источник в том же пункте дублируется на английском языке (References).

Источник на русском языке, перевод на английский доступен в метаданных статьи

1. **Журавлев С. С., Рудометов С. В., Окольнишников В. В., Шакиров С. Р.** Применение модельно-ориентированного проектирования к созданию АСУ ТП опасных промышленных объектов // Вестник НГУ. Серия: Информационные технологии. 2018. Т. 16, № 4. С. 56–67. DOI 10.25205/1818-7900-2018-16-4-56-67

Zhuravlev S. S., Rudometov S. V., Okolnishnikov V. V., Shakirov S. R. Model-Based Design Approach for Development Process Control Systems of Hazardous Industrial Facilities. *Vestnik NSU. Series: Information Technologies*, 2018, vol. 16, no. 4, p. 56–67. (in Russ.) DOI 10.25205/1818-7900-2018-16-4-56-67

Источник на английском языке. Оформляем согласно требованиям для References. Приводим только 1 раз.

2. **Telnov V. I.** Optimization of the Beam Crossing Angle at the ILC for E + e- and yy Collisions. *Journal of Instrumentation*, 2018, vol. 13, no. 03, p. P03020–P03020. DOI 10.1088/1748-0221/13/03/p03020

Метаданные источника доступны только на русском языке

3. **Жижимов О. Л., Федотов А. М., Шокин Ю. И.** Технологическая платформа массовой интеграции гетерогенных данных // Вестник НГУ. Серия: Информационные технологии. 2013. Т. 11, вып. 1. С. 24–41ю

Zhizhimov O. L., Fedotov A. M., Shokin Yu. I. Tekhnologicheskaya platforma massovoi integratsii geterogennykh dannykh [Technology Platform For the Mass Integration of Heterogeneous Data]. *Vestnik NSU. Series: Information Technologies*, 2013, vol. 11, no. 1, p. 24–41. (in Russ.)

Сведения об авторах

Последний раздел статьи – информация об авторе / авторах **на русском и английском языках**:

- ФИО полностью, ученая степень, ученое звание, должность, место работы, адрес места работы
- e-mail
- идентификаторы автора, такие как ORCID или ResearcherID (всем авторам рекомендуется использовать данные сервисы для ведения актуального списка своих публикаций)
- контактный телефон (не публикуется)

Если статья представляется на английском языке, необходимо приложить перевод на русский язык названия, аннотации, ключевых слов, сведений об авторе.

Доставка материалов

Материалы предоставляются в редакцию по электронной почте inftech@vestnik.nsu.ru.

Порядок рецензирования

Все статьи сначала проходят проверку на заимствование и только после этого отправляются на рецензирование. Редакционный совет не допускает к публикации материал, если имеется достаточно оснований полагать, что он является плагиатом.

Тип рецензирования статей – двухуровневое, одностороннее анонимное («слепое»).

Для каждой статьи редколлегией выбираются рецензенты, научная деятельность которых связана с темой представленного материала. Ответственный секретарь журнала обращается к ним с просьбой дать экспертную оценку статье либо помочь организовать рецензирование.

Рецензии для журнала «Вестник НГУ. Серия: Информационные технологии» составляются по единой схеме и подразумевают оценку по следующим критериям: соответствие тематике журнала, оригинальность и значимость результатов, качество изложения материала.

Заполненный бланк рецензии высылается на электронный адрес редакции. В зависимости от экспертных заключений статья может быть принята редакционным советом к опубликованию, рекомендована автору к доработке (с последующим повторным рецензированием либо без него) или отклонена (с предоставлением автору мотивированного отказа). Автору на электронный адрес высылается текст рецензии без указания ФИО рецензента и его контактных данных.

Все рецензии хранятся в редакции журнала не менее 5 лет. Редколлегия журнала обязуется при поступлении соответствующего запроса направлять копии рецензий в Министерство науки и высшего образования Российской Федерации.