ISSN 1818-1015
eISSN 2313-5417

2021 Volume 28 No 2

# MODELING AND ANALYSIS OF INFORMATION SYSTEMS

SCIENTIFIC JOURNAL

Start date of publication — 1999
Published quarterly

FOUNDER
P.G. Demidov Yaroslavl State University

EDITORIAL OFFICE
14 Sovetskaya str., Yaroslavl 150003, Russian Federation

Website: http://mais-journal.ru
E-mail: mais@uniyar.ac.ru
Phone: +7 (4852) 79-77-73

2021 Том 28 № 2

# МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ

НАУЧНЫЙ ЖУРНАЛ

Издается с 1999 года
Выходит 4 раза в год

УЧРЕДИТЕЛЬ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Ярославский государственный университет им. П. Г. Демидова»

РЕДАКЦИЯ
ул. Советская, 14, Ярославль, 150003, Российская Федерация
Website: http://mais-journal.ru
E-mail: mais@uniyar.ac.ru
Телефон: +7 (4852) 79-77-73

Свидетельство о регистрации СМИ ПИ № ФС 77–66186 от 20.06.2016 выдано Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций. Подписной индекс — 31907 в Объединенном каталоге «Пресса России». Технический редактор, компьютерная вёрстка – М. С. Каряева. Корректор английского текста – М. С. Комар. Подписано в печать 09.06.2021. Дата выхода в свет 30.06.2021. Формат 200×265 мм. Объем 94 с. Тираж 40 экз. Свободная цена. Заказ 022/021. Адрес типографии: ул. Советская, 14, оф. 109, Ярославль, 150003, Россия. Адрес издателя: Ярославский государственный университет им. П. Г. Демидова, ул. Советская, 14, Ярославль, 150003, Россия.
Содержание предназначено для детей старше 12 лет.

# Contents

# Содержание

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 2, 2021
journal homepage: www.mais-journal.ru

**ALGORITHMS**

# Identification Conditions for the Solvability of NP-complete Problems for the Class of Pre-fractal Graphs

A. V. Tymoshenko[1], R. A. Kochkarov[2], A. A. Kochkarov[2]

DOI: 10.18255/1818-1015-2021-2-126-135

[1]National Research University of Electronic Technology (MIET), 1 Shokin Square, Zelenograd 124498, Russia.
[2]Financial University under the Government of the Russian Federation, 49 Leningradsky Prospekt, Moscow 125993, Russia.

MSC2020: 37, 90
Research article
Full text in Russian

Received March 9, 2021
After revision April 27, 2021
Accepted May 12, 2021

Modern network systems (unmanned aerial vehicles groups, social networks, network production chains, transport and logistics networks, communication networks, cryptocurrency networks) are distinguished by their multi-element nature and the dynamics of connections between its elements. A number of discrete problems on the construction of optimal substructures of network systems described in the form of various classes of graphs are NP-complete problems. In this case, the variability and dynamism of the structures of network systems leads to an "additional" complication of the search for solutions to discrete optimization problems. At the same time, for some subclasses of dynamical graphs, which are used to model the structures of network systems, conditions for the solvability of a number of NP-complete problems can be distinguished. This subclass of dynamic graphs includes pre-fractal graphs.
The article investigates NP-complete problems on pre-fractal graphs: a Hamiltonian cycle, a skeleton with the maximum number of pendant vertices, a monochromatic triangle, a clique, an independent set. The conditions under which for some problems it is possible to obtain an answer about the existence and to construct polynomial (when fixing the number of seed vertices) algorithms for finding solutions are identified.

**Keywords:** NP-complete problems; pre-fractal graphs; discrete problems; solvability conditions

INFORMATION ABOUT THE AUTHORS

| | |
|---|---|
| Aleksandr Vasil'evich Tymoshenko<br>correspondence author | orcid.org/0000-0002-9791-142X. E-mail: u567ku78@gmail.com<br>Head of the Laboratory of Maintenance Systems and Diagnostics of Complex Radio Information Systems, Professor, Doctor of Science. |
| Rasul Ahmatovich Kochkarov | orcid.org/0000-0003-3186-3901. E-mail: rasul_kochkarov@mail.ru<br>Associate Professor, PhD in Economics. |
| Azret Ahmatovich Kochkarov | orcid.org/0000-0002-3232-5331. E-mail: akochkar@gmail.com<br>Associate Professor, PhD in Physics and Mathematics. |

**Funding:** The study was supported by a grant from the Russian Science Foundation No. 21-19-00481.

**For citation**: A. V. Tymoshenko, R. A. Kochkarov, and A. A. Kochkarov, "Identification Conditions for the Solvability of NP-complete Problems for the Class of Pre-fractal Graphs", *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 126-135, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 2, 2021
сайт журнала: www.mais-journal.ru

ALGORITHMS

# Выделение условий разрешимости NP-полных задач для класса предфрактальных графов

А. В. Тимошенко[1], Р. А. Кочкаров[2], А. А. Кочкаров[2]

DOI: 10.18255/1818-1015-2021-2-126-135

[1]Национальный исследовательский университет “Московский институт электронной техники”, Площадь Шокина, д. 1, г. Зеленоград, 124498 Россия.
[2]Финансовый университет при Правительстве РФ, Ленинградский просп., д. 49, г. Москва, 125993 (ГСП-3) Россия.

УДК 519.17
Научная статья
Полный текст на русском языке

Получена 9 марта 2021 г.
После доработки 27 апреля 2021 г.
Принята к публикации 12 мая 2021 г.

Современные сетевые системы (группы БПЛА, социальные сети, сетевые производственные цепочки, транспортно-логистические сети, сети связи, криптовалютные сети) отличаются многоэлементностью и динамикой связей между ее элементами. Ряд дискретных задач по построению оптимальных подструктур сетевых систем, описываемых в виде различных классов графов относятся к NP-полным задачам. При этом изменчивость и динамичность структур сетевых систем приводит к «дополнительному» усложнению поиска решения задач дискретной оптимизации. Вместе с тем для некоторых подклассов динамических графов, которыми моделируются структуры сетевых систем, можно выделить условия разрешимости ряда NP-полных задач. К такому подклассу динамических графов относятся предфрактальные графы.

В статье исследованы NP-полные задачи на предфрактальных графах: гамильтонов цикл, остов с максимальным числом висячих вершин, монохроматический треугольник, клика, независимое множество. Выделены условия, при которых для некоторых задач возможно получить ответ о существовании и построить полиномиальные (при фиксировании числа вершин затравки) алгоритмы поиска решений.

**Ключевые слова:** NP-полные задачи; предфрактальные графы; дискретные задачи; условия разрешимости

## ИНФОРМАЦИЯ ОБ АВТОРАХ

| | |
|---|---|
| Александр Васильевич Тимошенко<br>автор для корреспонденции | orcid.org/0000-0002-9791-142X. E-mail: u567ku78@gmail.com<br>начальник лаборатории систем технического обслуживания и диагностики сложных радиоинформационных комплексов, доктор технических наук, профессор. |
| Расул Ахматович Кочкаров | orcid.org/0000-0003-3186-3901. E-mail: rasul_kochkarov@mail.ru<br>доцент департамента анализа данных и машинного обучения, кандидат экономических наук, доцент. |
| Азрет Ахматович Кочкаров | orcid.org/0000-0002-3232-5331. E-mail: akochkar@gmail.com<br>доцент департамента анализа данных и машинного обучения, кандидат физ.-мат. наук. |

**Финансирование:** Исследование выполнено за счет гранта Российского научного фонда № 21-19-00481.

**Для цитирования**: A. V. Tymoshenko, R. A. Kochkarov, and A. A. Kochkarov, “Identification Conditions for the Solvability of NP-complete Problems for the Class of Pre-fractal Graphs”, *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 126-135, 2021.

## Введение

Количество прикладных задач в больших сетях только растет и с развитием направления анализа больших данных, увеличением числа абонентов и усложнением сетей, исследование задач на динамических графах большой размерности становится все более актуальным [1—4].

В работе [2] предложено описание динамических графов, проведен обзор известных методов их визуализации, предложены модифицированные алгоритмы с улучшенными характеристиками. Указанная последовательность соответствует термину траектории предфрактального (динамического) графа. С одной стороны, предлагаемые алгоритмы визуализации динамических графов применимы к визуализации предфрактальных графов как подклассу. С другой стороны, идентификация или распознавание социальных сетей в качестве предфрактальных графов позволит решать труднорешаемые задачи с улучшенной временной трудоемкостью, а также появляются возможности разработки параллельных алгоритмов или распараллеливания предложенных модернизированных алгоритмов.

В статье [3] предлагается метод генерации графа социальной сети, структура которого схожа с известными социальными сетями. Также авторами предложен алгоритм, учитывающий основные свойства социальной сети. Особенностью такого алгоритма стала зависимость от количества сообществ (или кластеров), а не не от количества пользователей. В терминологии предфрактальных графов в качестве сообществ выступают затравки (или блоки) разных рангов, при этом процедура порождения предфрактального графа позволяет переносить характеристики затравок на весь граф или по-другому строить граф с заранее заданными характеристиками.

Авторы [4] предложили эффективный алгоритм обнаружения и устранения коллизий в правилах межсетевого экрана в локальной сети. Древовидная структура рассмотрена в качестве структуры хранения правил политики безопасности. В соответствии с [5, 6] всякое дерево можно распознать в виде предфрактального графа. Адаптированный алгоритм [4] может быть применен в задачах выбора или конкурентной борьбы на предфрактальных и динамических графах.

## Определения и обозначения

В настоящей работе используется общепринятое обозначение и определение графа $G = (V, E)$ [7, 8]. Определение предфрактальных графов соответствует работам [5, 6]. *Затравкой* называется связный $n$-вершинный граф $H = (W, Q)$ с непомеченными вершинами $v \in W$. *Предфрактальный граф* обозначается в виде $G_L = (V_L, E_L)$, где $V_L$ — это множество вершин, а $E_L$ — множество ребер. В процессе построения предфрактального графа формируется траектория $G_1, G_2, ..., G_l, ..., G_L$. Построенный на шаге $l$ граф называется предфрактальным графом $G_l$ ранга $l$.

*Новыми* ребрами графа $G_L$ называются ребра ранга $L$, а остальные ребра *старыми* ребрами ранга $l$, где ранг $l = 1, 2, ..., L$ – это номер в траектории $G_1, G_2, ..., G_l, ..., G_L$. Далее для предфрактальных $(n, q, L)$-графов $G_L = (V_L, E_L)$ используется упрощенное обозначение $G_L$, где $q = |Q|$, $n = |W|$.

На рис. 1 представлена процедура замещения вершины затравкой (ЗВЗ). Вершины графа $G_1$ замещаются полной 3-вершинной затравкой $H$ с произвольной смежностью старых ребер: (а) малыми пунктирными окружностями обведены вершины, замещаемые затравкой; (б) средними пунктирными окружностями обведены затравки, замещающие вершины; (в) старые ребра графа $G_2$ выделены жирными линиями.

Важные характеристики предфрактального графа $G_L = (V_L, E_L)$ — количество его вершин $N = N(n, L) = |V_L|$ и ребер $M = M(n, q, L) = |E_L|$ соответственно:

$N = N(n, L) = n^L$;

$$M(n, q, L) = q(1 + n + n^2 + ... + n^{L-1}) = q\frac{n^L - 1}{n - 1}.$$

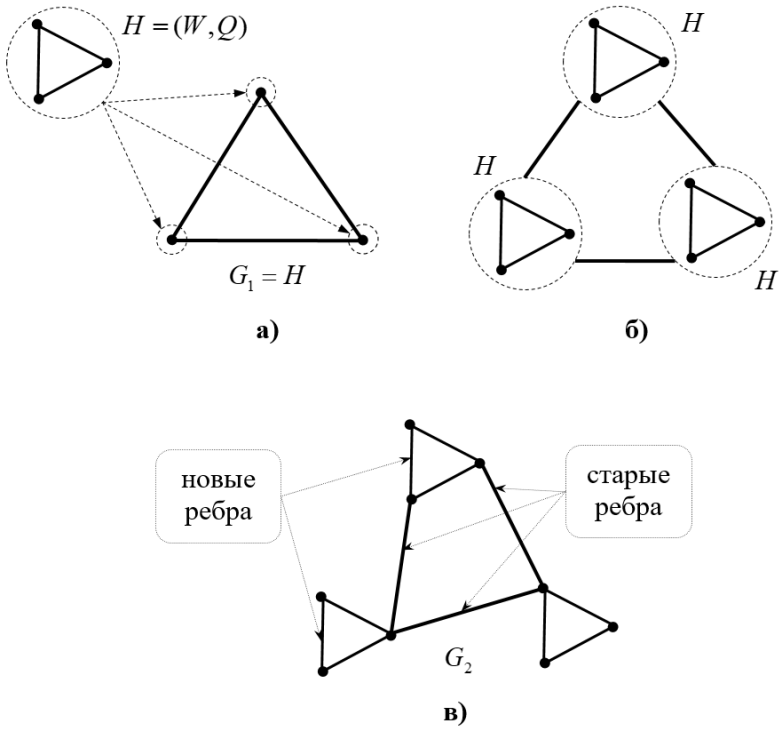



**Fig. 1.** The operation of replacing the vertices of the graph $G_1$ with a seed $H$

**Рис. 1.** Операция замещения вершин графа $G_1$ затравкой $H$

В процессе порождения используются различные условия смежности, основные из них: смежность старых ребер произвольная (рис. 1.в); смежность старых ребер сохраняется (рис. 2); старые ребра не пересекаются (рис. 3).

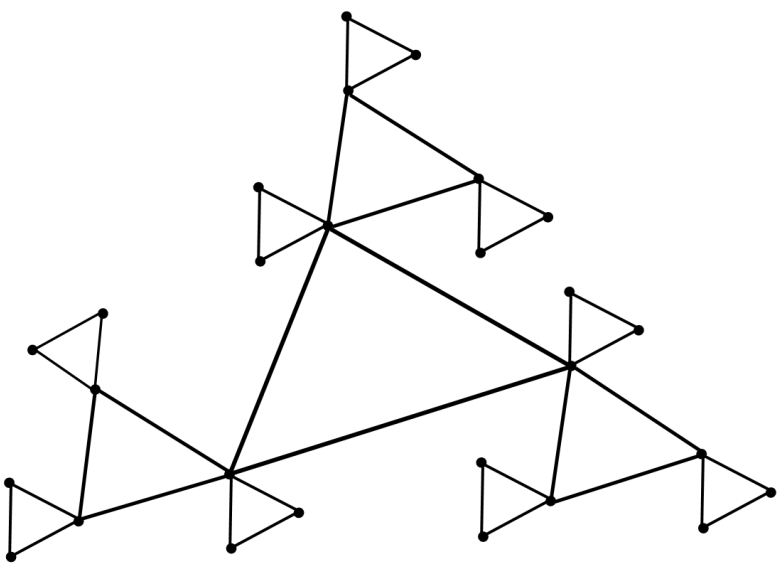


**Fig. 2.** Contiguity old edges preserved

**Рис. 2.** Смежность старых ребер сохраняется

Условие сохранения смежности старых ребер применяется для старых ребер всех рангов. В некоторых случаях оговаривается сохранение смежности старых ребер одного или нескольких рангов, например, сохранение смежности старых ребер с чередованием.

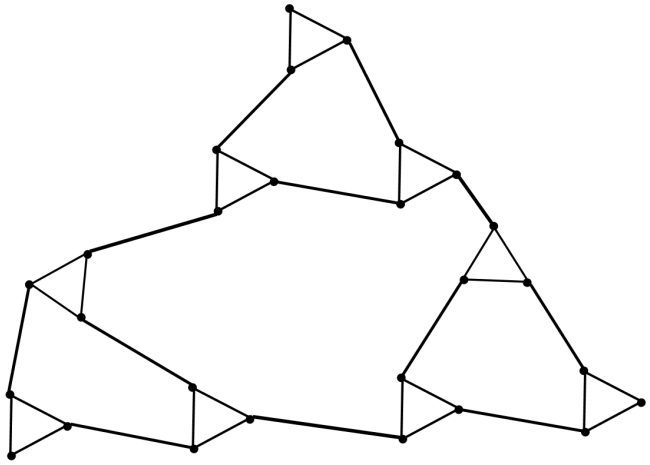


**Fig. 3.** Old edges are not crossed

**Рис. 3.** Старые ребра не пересекаются

Условие непересечения старых ребер, или, по-другому, отсутствие смежности применяется к старым ребрам всех рангов, в том числе старым ребрам разных рангов.

Ниже приводятся формулировки нескольких лемм, касающихся структурных особенностей предфрактальных графов.

ЛЕММА 1. Всякий предфрактальный граф $G_l, l \in [1, 2, ...L]$ можно построить с непересекающимися старыми ребрами.

ЛЕММА 2. Предфрактальный граф $G_l, l \in [1, 2, ...L]$, порожденный связной затравкой, является также связным.

Далее приводятся формулировки классических труднорешаемых задач, представленных в монографии [9]. После каждой задачи предлагается новая теорема для предфрактальных графов.

## 1. Гамильтонов цикл

ЗАДАЧА 1. Гамильтонов цикл.

УСЛОВИЕ. Задан граф $G = (V, E)$.

ВОПРОС. Имеется ли в $G$ гамильтонов цикл?

ТЕОРЕМА 1. Если на затравке $H$ имеется гамильтонов цикл, и между любой ее парой вершин имеется гамильтонова цепь, а старые ребра на предфрактальном графе не пересекаются, то на $G_l, l \in [1, 2, ...L]$, существует гамильтонов цикл.

ДОКАЗАТЕЛЬСТВО.

На графе $G_1$ существует гамильтонов цикл $C_1 = C_1^1$ по условию теоремы, так $G_1$ представляет собой затравку $H$. На следующем шаге ко всем вершинам $G_1$ применяется процедура ЗВЗ и формируется $G_2$ при этом старые ребра первого ранга не пересекаются. Или по-другому к вершинам гамильтонова цикла $C_1$ применяется процедура ЗВЗ и старые ребра первого ранга не пересекаются. Ребра первого ранга смежные в $C_1$ оказались разделенными затравками $H$. На каждой затравке выделяется гамильтонова цепь $C_s^2$ между ребрами первого ранга, которые были смежными в $C_1$. Таким образом, охвачены все вершины $G_2$ и построен гамильтонов цикл $C_2$.

Заместив вершины $C_l, l = 2, 3, ..., L - 1$ затравками $H$ и выделив на них гамильтоновы цепи $C_s^{l+1}$, $s = 1, 2, ..., n^{l-1}$, сформированы гамильтоновы циклы $C_{l+1}$ на предфрактальных графах $G_{l+1}$. Таким образом, для каждого предфрактального графа $G_l$ ранга $l \in [1, 2, ..., L]$ выделен гамильтонов цикл $C_l$.

Условие теоремы о непересечении старых ребер позволяет на каждом шаге разделять гамильтонов цикл подграф-затравками и достраивать недостающие части гамильтоновыми цепями. Обратное условие сохранения смежности старых ребер не позволяет построить гамильтонов цикл далее графа $G_1$, так как затравки $H$ на каждом шаге порождения склеиваются к существующим вершинам, и цикл проходит общие вершины не менее двух раз. ТЕОРЕМА ДОКАЗАНА.

СЛЕДСТВИЕ 1.1. Если смежность старых ребер предфрактального графа $G_l$, $l \in [1, 2, ...L]$ сохраняется, то на нем не существует гамильтонова цикла.

---

АЛГОРИТМ $\alpha_1$ ВЫДЕЛЕНИЯ ГАМИЛЬТОНОВА ЦИКЛА

ВХОД: взвешенный предфрактальный граф $G_L$.

ВЫХОД: гамильтонов цикл $C = (V, E_c)$.

ШАГ 1. Применяется процедура поиска гамильтонова цикла к $G_1$. На выходе процедуры получен гамильтонов цикл $C_1 = C_1^1$.

ДЛЯ ВСЕХ $l = 2, 3, ..., L$ ВЫПОЛНИТЬ:

ШАГ $l$. ДЛЯ ВСЕХ $s = 1, 2, ..., n^{l-1}$ ВЫПОЛНИТЬ:

ШАГ $l.s$. Для каждой $s$–ой затравки $l$–го ранга $H$ выполнить процедуру поиска гамильтоновой цепи $C_s^l$ между ребрами $(l-1)$–го ранга гамильтонового цикла $C_{l-1}$.

На выходе шага $l = 2, 3, ..., L$ выделен гамильтонов цикл $C_l$.

ПРОЦЕДУРА ГАМИЛЬТОНОВ ЦИКЛ

ВХОД: взвешенный граф $G = (V, E)$.

ВЫХОД: гамильтонов цикл $C_s = (V, E_s)$.

ПРОЦЕДУРА ГАМИЛЬТОНОВА ЦЕПЬ

ВХОД: взвешенный граф $G = (V, E)$.

ВЫХОД: гамильтонова цепь $C_s = (V, E_s)$.

---

СЛЕДСТВИЕ 1.2. При выполнении условий теоремы о существовании параметризованный алгоритм выделяет гамильтонов цикл на предфрактальном графе $G_l$ за время $O(c \cdot N)$, где $N = n^l$ и $c = (n-1)!$.

Используя переборный алгоритм, выделение гамильтонова цикла или цепи на затравке $H$ в худшем случае потребует выполнения $(n-1)!$ операций. Всего на предфрактальном графе $G_l$ присутствует $(n^l - 1)/(n-1) \leq n^l = N$ затравок.

ПРИМЕЧАНИЕ 1.1. В теореме 1 и следствии 1.1 рассматриваются два противоположных случая – старые ребра не пересекаются и смежность старых ребер сохраняется. При этом условие “старые ребра не пересекаются” означает, что старые ребра любых рангов не являются смежными между собой. Исследование остальных случаев является предметом дальнейших научных изысканий, в том числе в виде перечня теорем, которые покрывали бы всевозможные случаи порождения предфрактального графа.

ПРИМЕЧАНИЕ 1.2. В следствии 1.1 указанный алгоритм полиномиален относительно $N$ при фиксировании параметра $n$. Это примечание также верно для следствий 2.1 и 3.1.

ПРИМЕЧАНИЕ 1.3. Описание алгоритма $\alpha_1$ приведено в качестве примера. Алгоритмы разрабатываются в тесной связи с доказательствами соответствующих теорем. Далее для некоторых теорем указана трудоемкость алгоритмов без их описания.

## 2. Остов с максимальным числом висячих вершин

ЗАДАЧА 2. Остов с максимальным числом висячих вершин.

УСЛОВИЕ. Заданы граф $G = (V, E)$ и положительное целое число $K \leqslant |V|$.

ВОПРОС. Существует ли в графе $G$ остов, у которого не менее $K$ вершин степени 1?

ТЕОРЕМА 2. Если на затравке $H$ существует остов, у которого не менее $k \leq n$ висячих вершин и смежность старых ребер сохраняется, то на предфрактальном графе $G_l, l \in [1, 2, ...L]$ существует остов, у которого не менее $K$ висячих вершин, где $K = (k-1)n^{l-1}$.

ДОКАЗАТЕЛЬСТВО.

На графе $G_1$ существует остов $T_1 = T_1^1$, у которого не менее $k \leq n$ висячих вершин по условию теоремы, так $G_1$ представляет собой затравку $H$. На следующем шаге ко всем вершинам $G_1$ применяется процедура ЗВЗ и формируется $G_2$ с сохранением смежности старых ребер. Или по-другому к вершинам остова $T_1$ применяется процедура ЗВЗ с сохранением смежности старых ребер. Поскольку все вершины $T_1$ замещаются затравками, то ни одна такая вершина не является висячей в $G_2$. Выделяя остовы $T_s^2$ на затравках второго ранга и добавляя остов $T_1$, формируется остов $T_2$ на $G_2$. В остове $T_2$ висячими могут являться только висячие вершины в $T_s^2$. На любой затравке не менее $k \leq n$ висячих вершин. Остовы $T_s^2$ фактически склеиваются с вершинами $T_1$ одной из своих вершин. В худшем случае остов $T_s^2$ может склеиваться своей висячей вершиной, тогда количество висячих вершин остова $T_2$ в $G_2$ равно произведению числа висячих вершин без одной на затравке на число затравок второго ранга: $(k-1)n$.

Процедура порождения графов траектории $G_l, l = 2, 3, ..., L$ затравками $H$ и выделение на них остовов $T_s^l, s = 1, 2, ..., n^{l-1}$ формирует остовы $T_l$ на предфрактальных графах $G_l$. Таким образом, для каждого предфрактального графа $G_l$ ранга $l \in [1, 2, ...L]$ выделен остов $T_l$ с не менее $K = (k-1)n^{l-1}$ вершин. ТЕОРЕМА ДОКАЗАНА.

СЛЕДСТВИЕ 2.1. При выполнении условий теоремы о существовании параметризованный алгоритм выделяет на предфрактальном графе $G_l$ остов, у которого не менее $K = (k-1)n^{l-1}$ висячих вершин, за время $O(c \cdot N)$, где $N = n^l$ и $c = 2^n$.

Выделение остовных деревьев с максимальным числом висячих вершин на затравках может осуществляться любым известным алгоритмом, в худшем случае потребует выполнения $2^n$ операций. Всего на предфрактальном графе $G_l$ присутствует $(n^l - 1)/(n-1) \leq n^l = N$ затравок.

## 3. Монохроматический треугольник

ЗАДАЧА 3. Монохроматический треугольник.

УСЛОВИЕ. Задан граф $G = (V, E)$.

ВОПРОС. Существует ли разбиение множества $E$ на два непересекающихся множества $E_1$ и $E_2$, такие, что ни один из графов $G_1 = (V, E_1)$ или $G_2 = (V, E_2)$ не содержит треугольника?

ТЕОРЕМА 3.1. На предфрактальном графе $G_l, l \in [1, 2, ...L]$ не существует треугольников, состоящих из ребер соседних рангов.

ДОКАЗАТЕЛЬСТВО.

Рассматривается предфрактальный граф второго ранга $G_2$. Предположим, что существует треугольник, состоящий из двух ребер первого ранга и одного ребра второго ранга. Применим операцию стягивания затравок, обратную ЗВЗ, тогда два смежных ребра первого ранга стягиваются в кратное ребро, что противоречит определению предфрактального графа. Предположим, что существует треугольник, состоящий из двух ребер второго ранга и одного ребра первого ранга. Применив операцию стягивания затравок, два смежных ребра второго ранга стягиваются в вершину, а ребро первого ранга представляет собой петлю, что также противоречит определению предфрактального графа.

Рассматривается предфрактальный граф $l$-го ранга $G_l$. Предположим, что существует треугольник, состоящий из двух ребер $(l-1)$-го ранга и одного ребра $l$-го ранга. Применив операцию стягивания затравок $l$-го ранга, получим, что два смежных ребра $(l-1)$-го ранга стягиваются в кратное ребро, что противоречит определению предфрактального графа. Предположим, что существует треугольник, состоящий из двух ребер $l$-го ранга и одного ребра $(l-1)$-го ранга. Применяя операцию стягивания затравок $l$-го ранга, два смежных ребра $l$-го ранга стягиваются в вершину, а ребро

$(l-1)$-го ранга представляет собой петлю, что также противоречит определению предфрактального графа. Таким образом, на предфрактальном графе $G_l$ невозможно построить треугольник из ребер соседних рангов. ТЕОРЕМА ДОКАЗАНА.

ТЕОРЕМА 3.2. На предфрактальном графе $G_l, l \in [1, 2, ...L]$ не существует треугольников, состоящих из ребер разных рангов.

ДОКАЗАТЕЛЬСТВО.

На предфрактальном графе $G_l$ невозможно построить треугольник из ребер соседних рангов, как было доказано в предыдущей теореме.

Рассматривается предфрактальный граф $l$-го ранга $G_l$. Предположим, что существует треугольник, состоящий из двух ребер $l_1$-го ранга и одного ребра $l_2$-го ранга, $l_1 < l_2$ и $l_1, l_2 \in [1, 2, ..., L]$. Применив операцию стягивания затравок $l_2$-го ранга, получим, что два смежных ребра $l_1$-го ранга стягиваются в кратное ребро, что противоречит определению предфрактального графа. Предположим, что существует треугольник, состоящий из двух ребер $l_2$-го ранга и одного ребра $l_1$-го ранга, $l_1 < l_2$. Применяя операцию стягивания затравок $l_2$-го ранга, два смежных ребра $l_2$-го ранга стягиваются в вершину, а ребро $l_1$-го ранга представляет собой петлю, что также противоречит определению предфрактального графа. Теперь предположим, существует треугольник, состоящий из одного ребра $l_1$-го ранга, одного ребра $l_2$-го ранга и одного ребра $l_3$-го ранга $l_1 < l_2 < l_3$ и $l_1, l_2, l_3 \in [1, 2, ..., L]$. Применяя операцию стягивания затравок $l_3$-го ранга получим, что два смежных ребра $l_1$-го и $l_2$-го рангов стягиваются в кратное ребро, что противоречит процедуре порождения предфрактального графа и выводит из класса предфрактальных графов замкнутых относительно операции ЗВЗ. Таким образом, на предфрактальном графе $G_l$ невозможно построить треугольник из ребер разных рангов. ТЕОРЕМА ДОКАЗАНА.

ТЕОРЕМА 3.3. Если затравка $H$ – монохроматический треугольник, то предфрактальный граф $G_l, l \in [1, 2, ...L]$ также монохроматический треугольник.

ДОКАЗАТЕЛЬСТВО.

Предфрактальный граф $G_1$ является монохроматическим треугольником по условию теоремы, так как представляет собой затравку $H$. То есть $E_1$ можно разбить на два непересекающихся подмножества $E_1^1$ и $E_2^1$ , каждое из которых не содержит треугольников.

На графе второго ранга $G_2$ к графу $G_1$ добавляются затравки $H$, множество ребер каждой из них можно разбить на два непересекающихся подмножества $E_{s,1}^2$ и $E_{s,2}^2$. Объединяем все $E_{s,1}^2$ в одно множество $E'$, а все $E_{s,2}^2$ во второе множество $E''$. Новые треугольники после объединения не могут появиться, так как ребра затравок второго ранга не пересекаются. Добавляя $E_1^1$ к $E'$, а $E_2^1$ к $E''$ также новых треугольников не образуется, в силу выполнения теоремы о невозможности существования треугольников из ребер разных рангов. Таким образом, ребра $G_2$ разделены на два непересекающихся подмножества $E'$ и $E''$, не содержащих треугольников.

Рассматривается предфрактальный граф $l$-го ранга $G_l, l \in [2, 3, ...L]$, полученный из $G_{l-1}$ операцией ЗВЗ $H$. Ребра каждой затравки $l$-го ранга можно разбить на два непересекающихся подмножества $E_{s,1}^l$ и $E_{s,2}^l$. Объединяем все $E_{s,1}^l$ в одно множество $E'$, а все $E_{s,2}^l$ во второе множество $E''$. Новых треугольников после объединения не может появиться, так как ребра затравок $l$-го ранга не пересекаются. Добавляя $E_{s,1}^{l-1}$ к $E'$, а $E_{s,2}^{l-1}$ к $E''$ также новых треугольников не образуется, в силу выполнения теоремы о невозможности существования треугольников из ребер разных рангов. Получаем, что ребра $G_l$ разделены на два непересекающихся подмножества $E'$ и $E''$, не содержащих треугольников.

Таким образом, предложена процедура разделения множества ребер предфрактального графа $G_l, l \in [1, 2, ...L]$ на два непересекающихся подмножества $E'$ и $E''$, не содержащих треугольников. Процедура проводится пошагово от графа $G_1$ к $G_2$ и так далее до $G_l$ включительно. ТЕОРЕМА ДОКАЗАНА.

СЛЕДСТВИЕ 3.1. При выполнении условий теоремы о монохроматическом треугольнике параметризованный алгоритм разделяет множество ребер предфрактального графа $G_l, l \in [1, 2, ...L]$ на два непересекающихся подмножества, не содержащих треугольники, за время $O(c \cdot N)$, где $N = n^l$ и $c = 2^n$.

Используя переборный алгоритм, разделение множества ребер на затравке $H$ в худшем случае потребует выполнения $2^n$ операций. Всего на предфрактальном графе $G_l$ присутствует $(n^l - 1)/(n-1) \leq n^l = N$ затравок.

## 4. Клика

ЗАДАЧА 4. Клика.

УСЛОВИЕ. Заданы граф $G = (V, E)$ и положительное целое число $K \leqslant |V|$.

ВОПРОС. Верно ли, что $G$ содержит клику размера не менее $K$? Иными словами, существует ли подмножество $V' \subseteq V$ такое, что $|V'| \geqslant K$ и любые две вершины в $V'$ соединены ребром из $E$?

ТЕОРЕМА 4. Если на затравке $H$ существует клика размером не менее $k \leq n$, то на предфрактальном графе $G_l, l \in [1, 2, ...L]$ существует клика размером не менее $k$.

ДОКАЗАТЕЛЬСТВО.

На предфрактальном графе $G_l$ на затравках $l$-го ранга существуют клики размером не менее $k$ в силу условия теоремы.

Клики размера $k$, выделенные на затравках $l$-го ранга, нельзя увеличить за счет старых ребер предыдущих рангов. Клика состоит из треугольников и для того, чтобы перейти к клике размера $(k + 1)$, необходимо использовать ребра предыдущих рангов (ребра разных затравок одинакового ранга не пересекаются). Но построить треугольники из ребер разных рангов нельзя в силу ограничений теоремы о существовании треугольников, состоящих из ребер разных рангов. Таким образом, выбирая любую клику размера $k$ на затравках, невозможно увеличить ее до размера $(k + 1)$ на предфрактальном графе. ТЕОРЕМА ДОКАЗАНА.

## 5. Независимое множество

ЗАДАЧА 5. Независимое множество.

УСЛОВИЕ. Заданы граф $G = (V, E)$ и положительное целое число $K \leqslant |V|$.

ВОПРОС. Верно ли, что $G$ содержит независимое множество мощности не менее $K$? Иными словами, верно ли, что существует подмножество $V' \subseteq V$ такое, что $|V'| \geqslant K$ и никакие две вершины в $V'$ не соединены ребром из $E$?

ТЕОРЕМА 5. Если затравка $H$ содержит независимое множество мощности не менее $k \leq n$ и смежность старых ребер сохраняется, то предфрактальный граф $G_l, l \in [1, 2, ...L]$ содержит независимое множество мощности $K \leq kn^{l-1}$.

ДОКАЗАТЕЛЬСТВО.

По условию теоремы $G_1$ содержит независимое множество $V_1'$ мощности не менее $k$, так как представляет собой затравку $H$. Выделяя независимое множество $V_1'$ на $G_1$, смежные вершины перемещаются во множество $V_1''$. На следующем шаге из $G_1$ удаляются вершины $V_1''$ и перемещаются в $V_2''$, тогда в $G_2'$ все новые затравки второго ранга не пересекаются (в силу сохранения смежности старых ребер), так как вершины со смежными старыми ребрами первого ранга удалены. На урезанных затравках $G_2'$ выделяются независимые множества и сохраняются в $V_2'$, а все смежные вершины помещаются в $V_2''$. Множество $V_2''$ содержит в себе удаленные вершины на первом шаге $G_1$ и втором шаге $G_2$.

На следующем шаге из $G_3$ удаляются вершины $V_2''$ и перемещаются в $V_3''$, тогда в $G_3'$ все новые затравки третьего ранга не пересекаются, так как вершины со смежными старыми ребрами второго ранга удалены. На урезанных затравках $G_3'$ выделяются независимые множества и сохраняются в $V_3'$,

а все смежные вершины помещаются в $V_3''$. Множество $V_3''$ содержит в себе удаленные вершины на первом шаге $G_1$, втором шаге $G_2$ и третьем шаге $G_3$.

Процедура порождения графов траектории $G_l, l = 2, 3, ...L$ затравками $H$ формирует на предфрактальных графах $G_l$ независимые множества $V_l'$. Таким образом, для каждого предфрактального графа $G_l$ ранга $l \in [1, 2, ...L]$ выделено независимое множество $V_l'$ мощности $K \leq kn^{l-1}$. ТЕОРЕМА ДОКАЗАНА.

## Заключение

В работе рассмотрены и исследованы NP-полные задачи на одном из классов динамических графов – предфрактальных графах. По каждой из задач приведены условия разрешимости, при которых для некоторых подзадач возможно получить ответ о существовании и построить полиномиальные (при фиксировании числа вершин затравки) алгоритмы поиска решений. Ряд следствий обобщают полученные результаты, либо приводят условия – отсутствия решения. Например, для задачи выделения Гамильтонова цикла, приведено условие отсутствия решения на предфрактальном графе. Для некоторых задач приведены расчеты вычислительных сложностей поиска решений.

В данной работе приведено исследование малой части из числа известных NP-полных задач. Исследование всего набора задач и выделение условий разрешимости для класса динамических графов – предфрактальных графов, позволит по-сути говорить о выделении целого подкласса графов с условиями разрешимости NP-полных задач.

Разработка полиномиальных (при фиксировании числа вершин затравки) алгоритмов для динамических графов позволит решать с улучшенными характеристиками такие известные прикладные задачи как выделение подграфов в больших динамических сетях – выделение сообществ в социальных сетях; решение многокритериальных задач в транспортно-логистических системах большой размерности; поиск и выделение ddos-атак в криптовалютных системах и многие другие задачи.

## References


[1] A. Kochkarov, R. Kochkarov, and G. Malinetskii, “Issues of dynamic graph theory”, *Computational Mathematics and Mathematical Physics*, vol. 55, no. 9, pp. 1590–1596, 2015.

[2] S. Pupyrev and A. Tikhonov, “The Analysis of Complex Networks with Dynamic Graph Visualization”, *Modeling and Analysis of Information Systems*, vol. 17, no. 1, pp. 117–135, 2010.

[3] Y. Belov and S. Vovchok, “Generation of a Social Network Graph by Using Apache Spark”, *Modeling and Analysis of Information Systems*, vol. 23, no. 6, pp. 777–783, 2016.

[4] S. Morzhov and V. Sokolov, “An Effective Algorithm for Collision Resolution in Security Policy Rules”, *Modeling and Analysis of Information Systems*, vol. 26, no. 1, pp. 75–89, 2019.

[5] A. M. Kochkarov, V. A. Perepelitsa, and I. V. Sergienko, *Recognition of fractal graphs. Algorithmic approach*. RAS SAO, 1998.

[6] R. A. Kochkarov, “Problems of multicriteria optimization on multi-weighted prefractal graphs”, no. 17, pp. 319–328, 2014.

[7] F. Harary, *Graph theory*. Addison-Wesley Pub. Co., 1969.

[8] M. A. Iordanskii, “Constructive Classification of Graphs”, *Modeling and Analysis of Information Systems*, vol. 19, no. 4, pp. 144–153, 2012.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 2, 2021
journal homepage: www.mais-journal.ru

**ALGORITHMS**

# Computational Analysis of Quantitative Characteristics of some Residual Properties of Solvable Baumslag–Solitar Groups

E. A. Tumanova[1]

DOI: 10.18255/1818-1015-2021-2-136-145

[1]Ivanovo State University, 39 Ermak str., Ivanovo 153025, Russia.

MSC2020: Primary 68R05, 20E26, 62-07. Secondary 68Q25, 20F05, 62-04
Research article
Full text in Russian

Received April 26, 2021
After revision May 28, 2021
Accepted June 2, 2021

Let $G_k$ be defined as $G_k = \langle a, b;\ a^{-1}ba = b^k \rangle$, where $k \neq 0$. It is known that, if $p$ is some prime number, then $G_k$ is residually a finite $p$-group if and only if $p \mid k-1$. It is also known that, if $p$ and $q$ are primes not dividing $k-1$, $p < q$, and $\pi = \{p, q\}$, then $G_k$ is residually a finite $\pi$-group if and only if $(k, q) = 1$, $p \mid q-1$, and the order of $k$ in the multiplicative group of the field $\mathbb{Z}_q$ is a $p$-number. This paper examines the question of the number of two-element sets of prime numbers that satisfy the conditions of the last criterion. More precisely, let $f_k(x)$ be the number of sets $\{p, q\}$ such that $p < q$, $p \nmid k-1$, $q \nmid k-1$, $(k, q) = 1$, $p \mid q-1$, the order of $k$ modulo $q$ is a $p$-number, and $p$, $q$ are chosen among the first $x$ primes. We state that, if $2 \leq |k| \leq 10000$ and $1 \leq x \leq 50000$, then, for almost all considered $k$, the function $f_k(x)$ can be approximated quite accurately by the function $\alpha_k x^{0.85}$, where the coefficient $\alpha_k$ is different for each $k$ and $\{\alpha_k \mid 2 \leq |k| \leq 10000\} \subseteq (0.28;\ 0.31]$. We also investigate the dependence of the value $f_k(50000)$ on $k$ and propose an effective algorithm for checking a two-element set of prime numbers for compliance with the conditions of the last criterion. The results obtained may have applications in the theory of computational complexity and algebraic cryptography.

**Keywords:** Baumslag–Solitar groups; residual $\pi$-finiteness; function approximation; analysis of algorithms

INFORMATION ABOUT THE AUTHORS

Elena Alexandrovna Tumanova
correspondence author

orcid.org/0000-0002-6193-9834. E-mail: helenfog@bk.ru
Associate Professor, Ph. D. in Mathematics.

**For citation**: E. A. Tumanova, "Computational Analysis of Quantitative Characteristics of some Residual Properties of Solvable Baumslag–Solitar Groups", *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 136-145, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 2, 2021
сайт журнала: www.mais-journal.ru



# Вычислительный анализ количественных характеристик некоторых аппроксимационных свойств разрешимых групп Баумслага–Солитэра

Е. А. Туманова[1]

DOI: 10.18255/1818-1015-2021-2-136-145

[1]Ивановский государственный университет, ул. Ермака, д. 39, г. Иваново, 153025 Россия.

УДК 004.421, 512.54, 519.25, 519.65
Научная статья
Полный текст на русском языке

Получена 26 апреля 2021 г.
После доработки 28 мая 2021 г.
Принята к публикации 2 июня 2021 г.

Пусть $G_k = \langle a, b;\ a^{-1}ba = b^k \rangle$, где $k \neq 0$. Известно, что если $p$ — некоторое простое число, то группа $G_k$ аппроксимируется конечными $p$-группами тогда и только тогда, когда $p \mid k - 1$. Известно также, что если $p$ и $q$ — простые числа, не делящие $k - 1$, $p < q$ и $\pi = \{p,\ q\}$, то группа $G_k$ аппроксимируется конечными $\pi$-группами тогда и только тогда, когда $(k, q) = 1$, $p \mid q - 1$ и порядок числа $k$ в мультипликативной группе поля $\mathbb{Z}_q$ является $p$-числом. В настоящей статье исследуется вопрос о количестве двухэлементных множеств простых чисел, удовлетворяющих условиям последнего критерия. Более точно, пусть $f_k(x)$ — количество множеств $\{p,\ q\}$ таких, что $p < q$, $p \nmid k - 1$, $q \nmid k - 1$, $(k, q) = 1$, $p \mid q - 1$, порядок $k$ по модулю $q$ является $p$-числом и $p$, $q$ выбираются среди первых $x$ простых чисел. Установлено, что если $2 \leq |k| \leq 10000$ и $1 \leq x \leq 50000$, то почти для всех рассматриваемых $k$ функция $f_k(x)$ может быть достаточно точно приближена функцией $\alpha_k x^{0,85}$, где коэффициент $\alpha_k$ — свой для каждого $k$ и $\{\alpha_k \mid 2 \leq |k| \leq 10000\} \subseteq (0,28;\ 0,31]$. Также исследована зависимость величины $f_k(50000)$ от $k$ и предложен эффективный алгоритм проверки двухэлементного множества простых чисел на соответствие условиям последнего критерия. Полученные результаты могут иметь приложения в теории сложности вычислений и алгебраической криптографии.

**Ключевые слова:** группы Баумслага–Солитэра; аппроксимируемость конечными $\pi$-группами; аппроксимация функций; анализ алгоритмов

ИНФОРМАЦИЯ ОБ АВТОРАХ

Елена Александровна Туманова
автор для корреспонденции
orcid.org/0000-0002-6193-9834. E-mail: helenfog@bk.ru
доцент, канд. физ.-мат. наук.

**Для цитирования**: E. A. Tumanova, “Computational Analysis of Quantitative Characteristics of some Residual Properties of Solvable Baumslag–Solitar Groups”, *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 136-145, 2021.

## 1. Некоторые аппроксимационные свойства рассматриваемых групп

Следуя [1], для каждого ненулевого целого числа $k$ обозначим через $G_k$ группу, которая задается представлением $G_k = \langle a, b;\ a^{-1}ba = b^k \rangle$. Она входит в хорошо известное семейство групп Баумслага–Солитэра $\mathrm{BS}(m, n) = \langle a, b;\ a^{-1}b^m a = b^n \rangle$, где $m$ и $n$ — ненулевые целые числа.

Изучение аппроксимационных свойств группы $G_k$ было начато Г. Баумслагом и Д. Солитэром в [2]. Напомним, что группа $G$ называется *аппроксимируемой классом групп* $\mathcal{C}$ (*$\mathcal{C}$-аппроксимируемой*), если для любого неединичного элемента $g \in G$ существует гомоморфизм $\varphi$ группы $G$ на группу из класса $\mathcal{C}$ такой, что $g\varphi \neq 1$. В [2, 3] установлено, что группа $G_k$ финитно аппроксимируема, то есть аппроксимируема классом всех конечных групп. Исследование аппроксимируемости этой группы конечными группами было продолжено Д. И. Молдаванским. В [4] он доказал следующий критерий аппроксимируемости группы $G_k$ классом $\mathcal{F}_p$ всех конечных $p$-групп, где $p$ — некоторое простое число.

**Теорема 1.** [4, теорема 3] *Пусть $p$ — простое число. Группа $G_k$ $\mathcal{F}_p$-аппроксимируема тогда и только тогда, когда $p$ делит $k - 1$.*

В [5] была рассмотрена аппроксимируемость группы $G_k$ классом $\mathcal{F}_\pi$ всех конечных $\pi$-групп, где $\pi$ — некоторое множество простых чисел. Напомним, что целое число называется *$\pi$-числом*, если все его простые делители принадлежат множеству $\pi$, периодическая группа называется *$\pi$-группой*, если порядки всех ее элементов являются $\pi$-числами. Приводимая далее теорема 2 дает критерий аппроксимируемости группы $G_k$ конечными $\pi$-группами для произвольного множества $\pi$ простых чисел. Поясним, что в ее формулировке и всюду далее под порядком целого числа $x$ по модулю целого числа $y$ понимается порядок числа $x$ в мультипликативной группе кольца $\mathbb{Z}_y$.

**Теорема 2.** [5, теорема 1] *Пусть $\pi$ — произвольное множество простых чисел. Группа $G_k$ $\mathcal{F}_\pi$-аппроксимируема тогда и только тогда, когда существует $\pi$-число $s > 1$, взаимно простое с $k$, порядок по модулю которого числа $k$ также является $\pi$-числом.*

Критерий, содержащийся в теореме 2, не является конструктивным, т. е. не дает алгоритма, позволяющего для заданных числа $k$ и множества $\pi$ проверить, аппроксимируется ли группа $G_k$ классом $\mathcal{F}_\pi$. Такой алгоритм существует для двухэлементного множества простых чисел и предоставляется приводимой далее теоремой 3. Отметим, что в этой теореме рассматриваются только пары простых чисел $p$ и $q$, каждое из которых не делит $k - 1$, поскольку в противном случае группа $G_k$ аппроксимируется классом $\mathcal{F}_p$ или $\mathcal{F}_q$ согласно теореме 1.

**Теорема 3.** [5, теорема 2] *Пусть $p$ и $q$ — простые числа, удовлетворяющие условиям*

*a*) $p < q$;

*b*) $p \nmid k - 1$;

*c*) $q \nmid k - 1$.

*Пусть также $\pi = \{p, q\}$. Группа $G_k$ $\mathcal{F}_\pi$-аппроксимируема тогда и только тогда, когда*

1) $(k, q) = 1$;

2) $p \mid q - 1$;

3) *порядок $k$ по модулю $q$ является $p$-числом.*

Хотя теорема 3 имеет эффективно проверяемое условие, она не дает ответа на вопрос, для всех ли значений $k$ существует пара простых чисел $(p, q)$, удовлетворяющая ее условию. Нетрудно проверить, что при $k = \pm 1$ такой пары нет. При $|k| > 1$ наличие пары простых чисел с требуемыми свойствами устанавливает

**Теорема 4.** [5, теорема 3] *Если $|k| > 1$, то для любого простого числа $p$, не делящего $k - 1$, найдется простое число $q$, не делящее $k - 1$ и такое, что группа $G_k$ $\mathcal{F}_\pi$-аппроксимируема, где $\pi = \{p, q\}$.*

Теорема 4 оставляет открытым вопрос о количестве двухэлементных множеств простых чисел, удовлетворяющих условиям теоремы 3. В данной работе этот вопрос исследуется при условии, что $2 \le |k| \le 10000$ и элементы множества выбираются из первых 50000 простых чисел. Указанные ограничения обусловлены временем работы программы, перечисляющей двухэлементные множества простых чисел и выбирающей из них те, для которых справедливы условия теоремы 3. Попытка дальнейшего увеличения диапазона рассматриваемых простых чисел приводит к несоразмерному росту времени выполнения расчетов исходных данных для проведения исследования; подробнее об этом см. в разделе 4.

## 2. Зависимость от $k$ числа двухэлементных множеств, удовлетворяющих условиям теоремы 3

Пусть $|k| > 1$ и $n \ge 1$. Будем использовать следующие обозначения:

$\mathcal{P}(n)$ — множество, составленное из первых $n$ простых чисел;

$S_k(n)$ — множество пар простых чисел $(p, q)$, удовлетворяющих условиям $a$–$c$, 1–3 теоремы 3 и таких, что $p, q \in \mathcal{P}(n)$;

$f_k(n)$ — мощность множества $S_k(n)$.

Положим также

$$\mathcal{K} = \{-10000; -2\} \cup \{2; 10000\}, \quad \mathfrak{C}(k) = f_k(50000), \quad \mathfrak{c}(k) = f_k(4).$$

Посредством явного вычисления элементов множеств $S_k(50000)$ при $2 \le |k| \le 10000$ было установлено, что значения $\mathfrak{C}(k)$ меняются в достаточно широких пределах. Если рассматривать функцию $\mathfrak{C} : \mathcal{K} \longrightarrow \mathbb{Z}$ как случайную величину, то в предположении равновероятности всех элементарных событий имеем

$$M(\mathfrak{C}) \approx 2940,57; \quad \sigma(\mathfrak{C}) \approx 280,02.$$

Можно выделить некоторые значения $k$, для которых величина $\mathfrak{C}(k)$ существенно отличается от $M(\mathfrak{C})$. Неравенство $\mathfrak{C}(k) < \frac{1}{2}M(\mathfrak{C})$ выполняется для 76 элементов множества $\mathcal{K}$, причем все эти числа отрицательны. Среди них следует особо отметить те значения $k$, для которых $\mathfrak{C}(k) < \frac{1}{4}M(\mathfrak{C})$: $-256$; $-16$; $-6561$; $-4096$; $-576$; $-9216$; $-2916$. Все положительные значения $k \in \mathcal{K}$ удовлетворяют соотношению $\mathfrak{C}(k) > 0,73M(\mathfrak{C})$.

В случае с превышением величиной $\mathfrak{C}(k)$ среднего значения $M(\mathfrak{C})$ картина складывается прямо противоположная. Неравенство $\mathfrak{C}(k) > 2M(\mathfrak{C})$ выполняется для 37 значений $k$, и все они положительны. Нельзя не обратить внимание на $k = 4096$, так как $\mathfrak{C}(4096) = 11878 > 4M(\mathfrak{C})$ и $\mathfrak{C}(k) < 3M(\mathfrak{C})$ для всех остальных $k \in \mathcal{K}$. Все отрицательные значения $k \in \mathcal{K}$ удовлетворяют соотношению $\mathfrak{C}(k) < 1,62M(\mathfrak{C})$.

Точки, в которых функция $\mathfrak{C}(k)$ принимает 20 самых больших и 20 самых маленьких значений, приводятся в табл. 1. Анализируя ее содержимое, логично предположить, что значение функции $\mathfrak{C}(k)$ существенным образом связано с разложением числа $k$ на простые множители. Изучение такой зависимости может стать предметом дальнейших исследований.

Еще один естественным образом возникающий вопрос — можно ли довольно быстро вручную найти пару простых чисел, удовлетворяющих условиям теоремы 3. Для ответа на него была исследована функция $\mathfrak{c}(k)$, соответствующая случаю, когда простые числа выбираются из множества $\mathcal{P}(4) = \{2, 3, 5, 7\}$.

Установлено, что для 10000 из 19998 рассмотренных значений $k$ справедливо неравенство $\mathfrak{c}(k) > 0$, причем среди этих 10000 значений $k$ положительных и отрицательных практически поровну: 4998 положительных и 5002 отрицательных. Таким образом, примерно для половины элементов множества $\mathcal{K}$ искомую пару простых чисел можно быстро найти подбором. Кроме этого, установлено, что $\mathfrak{c}(k) \le 3$ для любого $k \in \mathcal{K}$. Количества элементов множества $\mathcal{K}$, для которых функция $\mathfrak{c}(k)$ принимает значения 0, 1, 2, 3, приведены в табл. 2.

**Table 1.** The largest and smallest values of the function $\mathfrak{C}(k)$

**Таблица 1.** Наибольшие и наименьшие значения функции $\mathfrak{C}(k)$

| $k$ | $\mathfrak{C}(k)$ | $\mathfrak{C}(k)/M(\mathfrak{C})$ | $k$ | $\mathfrak{C}(k)$ | $\mathfrak{C}(k)/M(\mathfrak{C})$ |
|---|---|---|---|---|---|
| −256 | 267 | $\approx 0,091$ | 225 | 6038 | $\approx 2,053$ |
| −16 | 481 | $\approx 0,164$ | 441 | 6049 | $\approx 2,057$ |
| −6561 | 570 | $\approx 0,194$ | 100 | 6050 | $\approx 2,057$ |
| −4096 | 693 | $\approx 0,236$ | 3600 | 6067 | $\approx 2,063$ |
| −576 | 727 | $\approx 0,247$ | 9216 | 6243 | $\approx 2,123$ |
| −9216 | 732 | $\approx 0,249$ | 576 | 6254 | $\approx 2,127$ |
| −2916 | 733 | $\approx 0,249$ | 36 | 6277 | $\approx 2,135$ |
| −36 | 739 | $\approx 0,251$ | 2916 | 6296 | $\approx 2,141$ |
| −2401 | 919 | $\approx 0,313$ | 1024 | 6408 | $\approx 2,179$ |
| −625 | 935 | $\approx 0,318$ | 81 | 6918 | $\approx 2,353$ |
| −10000 | 959 | $\approx 0,326$ | 1296 | 6990 | $\approx 2,377$ |
| −81 | 1035 | $\approx 0,352$ | 625 | 7175 | $\approx 2,440$ |
| −1296 | 1052 | $\approx 0,358$ | 10000 | 7190 | $\approx 2,445$ |
| −100 | 1169 | $\approx 0,398$ | 2401 | 7271 | $\approx 2,473$ |
| −1600 | 1187 | $\approx 0,404$ | 64 | 7429 | $\approx 2,526$ |
| −8100 | 1194 | $\approx 0,406$ | 6561 | 7947 | $\approx 2,703$ |
| −8427 | 1228 | $\approx 0,418$ | 16 | 8206 | $\approx 2,791$ |
| −1587 | 1249 | $\approx 0,425$ | 729 | 8306 | $\approx 2,825$ |
| −9075 | 1268 | $\approx 0,431$ | 256 | 8660 | $\approx 2,945$ |
| −4107 | 1269 | $\approx 0,432$ | 4096 | 11878 | $\approx 4,039$ |

**Table 2.** Frequencies of the values of the function $\mathfrak{c}(k)$

**Таблица 2.** Частоты встречаемости значений функции $\mathfrak{c}(k)$

| $x$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $\operatorname{card}\{k \in \mathcal{K} \mid \mathfrak{c}(k) = x\}$ | 9998 | 6285 | 2858 | 857 |

## 3. Зависимость числа двухэлементных множеств, удовлетворяющих условиям теоремы 3, от количества рассматриваемых простых чисел

В данном разделе исследуется вопрос о том, как меняется количество пар простых чисел, удовлетворяющих условию теоремы 3, т. е. величина $f_k(n)$, в зависимости от количества $n$ рассматриваемых простых чисел при фиксированном $k$. Можно ли одной из элементарных функций довольно хорошо приблизить такую зависимость? Из результатов предыдущего параграфа ясно, что при различных $k$ функции $f_k(x)$ имеют существенно различающиеся значения. Но влияет ли $k$ на характер зависимости или функции $f_k(x)$ отличаются друг от друга лишь некоторым множителем?

Чтобы ответить на эти вопросы, вычисленные значения функций $f_k(x)$ были визуализированы для некоторых $k$ (см. рис. 1). В результате возникло предположение о том, что при различных $k$ функции $f_k(x)$ имеют один и тот же характер зависимости, похожий на логарифмический или степенной (с показателем степени, меньшим 1). Первые же попытки аппроксимации показали, что логарифмическая функция не подходит. Поэтому далее было решено искать приближенную функцию в виде $g_k(x) = \alpha x^\beta$, где коэффициенты $\alpha$ и $\beta$, вообще говоря, зависят от $k$.

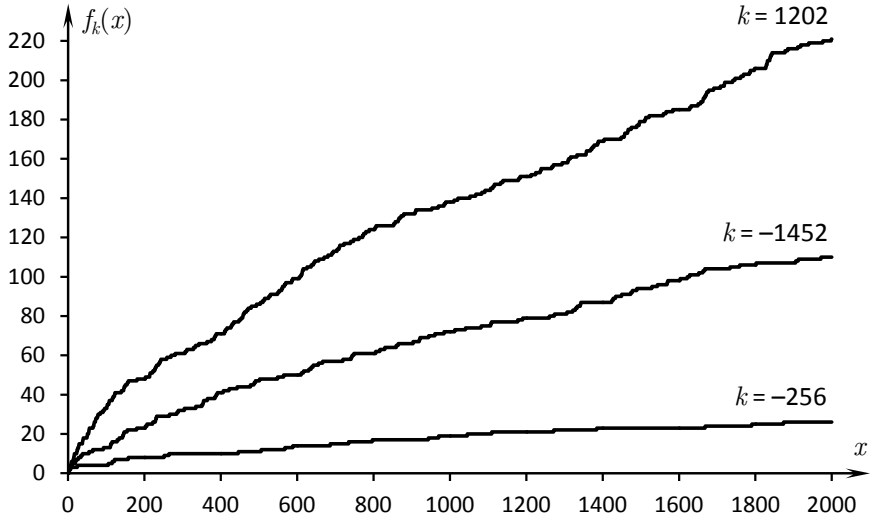



**Fig. 1.** The graph of the function $f_k(x)$ on the set $\{1, 2, \ldots, 2000\}$ for different $k$

**Рис. 1.** График функции $f_k(x)$ на множестве $\{1, 2, \ldots, 2000\}$ при различных $k$

При заданных $k$ и $\beta$ поиск множителя $\alpha$ выполнялся, исходя из условия

$$\sum_{x=1}^{50000} \left(1 - \frac{\alpha x^{\beta}}{\tilde{f}_k(x)}\right)^2 \to \min,$$

где

$$\tilde{f}_k(x) = \begin{cases} f_k(x), & \text{если } f_k(x) \neq 0, \\ 1, & \text{если } f_k(x) = 0, \end{cases}$$

откуда

$$\alpha = \left(\sum_{x=1}^{50000} x^{\beta}\right) \Bigg/ \left(\sum_{x=1}^{50000} \frac{x^{2\beta}}{\tilde{f}_k(x)}\right).$$

Определение $\beta$ осуществлялось двумя способами.

*Способ 1.* Перебор для каждого $k \in \mathcal{K}$ всех значений из множества $\mathcal{B}$, содержащего числа от $0,7$ до $0,99$ с шагом $0,01$. Минимальный элемент множества $\mathcal{B}$ был найден в ходе предварительных расчетов с меньшим количеством точек, в которых вычислялись функции $f_k(x)$.

При фиксированном $k \in \mathcal{K}$ для каждого $b \in \mathcal{B}$ определялись соответствующие ему значения

$$\alpha(b) = \left(\sum_{x=1}^{50000} x^{b}\right) \Bigg/ \left(\sum_{x=1}^{50000} \frac{x^{2b}}{\tilde{f}_k(x)}\right), \qquad \delta(b) = \sum_{x=1}^{50000} \left(1 - \frac{\alpha(b) x^{b}}{\tilde{f}_k(x)}\right)^2,$$

после чего выбирался тот элемент $b \in \mathcal{B}$, для которого значение $\delta(b)$ оказалось минимальным. Этот элемент далее обозначается через $\beta_k$.

*Способ 2.* Использование для всех $k \in \mathcal{K}$ одного и того же элемента $b \in \mathcal{B}$.

Выбирался элемент $b \in \mathcal{B}$, наиболее близкий к среднему значению чисел $\beta_k$ ($k \in \mathcal{K}$), полученных первым способом. Этот элемент далее обозначается через $\beta_{avg}$.

Отметим, что определение $\beta$ с большей точностью не производилось, поскольку исходная задача состояла лишь в описании общего вида функций $f_k(x)$, а не в получении возможности отыскания

их значений с помощью аппроксимирующей функции $g_k(x)$. При этом найденное таким путем приближение оказалось вполне удовлетворительным (см. ниже).

Возможность выбора коэффициента $\beta$ вторым способом обеспечивается особенностью распределения чисел $\beta_k$ ($k \in \mathcal{K}$). Пусть функции $\beta : \mathcal{K} \to \mathcal{B}$ и $\eta : \mathcal{B} \to \mathbb{Z}$ определены следующим образом: $\beta(k) = \beta_k$, $\eta(b) = \operatorname{card}\{k \in \mathcal{K} \mid \beta(k) = b\}$. Значения функции $\eta$, представленные в табл. 3, позволяют сделать предположение о том, что функция $\beta$, рассматриваемая как дискретная случайная величина, имеет нормальное распределение. Проверим для нее выполнение правила «трех сигм».

Имеем $M(\beta) \approx 0,8531$ и $\sigma(\beta) \approx 0,0206$. Случайная величина $\beta(k)$ содержится в интервале $\big(M(\beta) - 3\sigma(\beta);\ M(\beta) + 3\sigma(\beta)\big)$ с вероятностью $\approx 0,9954$, в интервале $\big(M(\beta) - 2\sigma(\beta);\ M(\beta) + 2\sigma(\beta)\big)$ — с вероятностью $\approx 0,9531$, в интервале $\big(M(\beta) - \sigma(\beta);\ M(\beta) + \sigma(\beta)\big)$ — с вероятностью $\approx 0,6688$. Правило «трех сигм» не выполнено, но полученные вероятности лишь немного отличаются от необходимых значений, поэтому можно считать, что распределение случайной величины $\beta(k)$ близко к нормальному.

**Table 3.** Nonzero values of the function $\eta$ **Таблица 3.** Ненулевые значения функции $\eta$

| $b$ | 0,71 | 0,75 | 0,76 | 0,77 | 0,78 | 0,79 | 0,80 | 0,81 | 0,82 | 0,83 | 0,84 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\eta(b)$ | 1 | 1 | 2 | 4 | 14 | 16 | 71 | 337 | 1030 | 2297 | 3426 |
| $b$ | | 0,85 | 0,86 | 0,87 | 0,88 | 0,89 | 0,90 | 0,91 | 0,92 | 0,93 | 0,94 |
| $\eta(b)$ | | 3886 | 3457 | 2605 | 1566 | 793 | 323 | 115 | 37 | 11 | 6 |

Отметим, что два самых маленьких значения функции $\beta$ соответствуют элементам множества $\mathcal{K}$ из табл. 1. Так, $\beta(k) = 0,71$ при $k = -256$ и $\beta(k) = 0,75$ при $k = -2916$. Что касается максимума функции $\beta(k) = 0,94$, то здесь ситуация обратная — в пяти точках из шести, где он достигается, случайная величина $\mathfrak{C}$ принимает значение, близкое к $M(\mathfrak{C})$. Исключение составляет $k = -1849$, для которого $\mathfrak{C}(k) = 1630$. Таким образом, явной взаимосвязи между точками, в которых функции $\mathfrak{C}$ и $\beta$ принимают значения, близкие к предельным, не наблюдается.

При использовании способов 1 и 2 для каждого $k \in \mathcal{K}$ были найдены два значения коэффициента $\alpha$; обозначим их через $\alpha_k^{(1)}$ и $\alpha_k^{(2)}$ соответственно и отметим, что

$$\big\{\alpha_k^{(1)} \bigm| k \in \mathcal{K}\big\} \subseteq \big[0,064;\ 0,775\big], \quad \big\{\alpha_k^{(2)} \bigm| k \in \mathcal{K}\big\} \subseteq \big[0,028;\ 1,186\big].$$

Чтобы описать распределение значений $\alpha_k^{(1)}$ и $\alpha_k^{(2)}$ более точно, введем функции

$$\mathfrak{A}^{(i)} : \mathcal{A} \to \mathbb{Z}, \quad \mathfrak{A}^{(i)}(a) = \operatorname{card}\big\{k \in \mathcal{K} \bigm| a - 0,01 < \alpha_k^{(i)} \le a\big\},$$

где $i \in \{1;\ 2\}$, $\mathcal{A} = \{0,01;\ 0,02;\ \ldots;\ 0,99;\ 1\}$. Графики этих функций в логарифмическом масштабе на множестве $\mathcal{A}' = \{0,1;\ 0,11;\ \ldots;\ 0,59;\ 0,6\}$ представлены на рис. 2. Отметим, что $\mathfrak{A}^{(1)}(a) \le 3$ и $\mathfrak{A}^{(2)}(a) \le 7$ при $a \notin \mathcal{A}'$.

Анализируя значения функций $\mathfrak{A}^{(1)}$ и $\mathfrak{A}^{(2)}$, можно сделать следующие выводы.

1. Подавляющее большинство (18997 из 19998) коэффициентов $\alpha_k^{(2)}$ принадлежит промежутку $(0,28;\ 0,31]$, при этом 12336 из них содержится в промежутке $(0,29;\ 0,30]$. Как следствие, при аппроксимации всех функций $f_k(x)$ можно попытаться использовать один и тот же коэффициент $\alpha$, близкий к среднему значению чисел $\alpha_k^{(2)}$ ($k \in \mathcal{K}$).

2. Функция $\mathfrak{A}^{(1)}$ имеет несколько локальных максимумов, близких по величине и не превосходящих 2535. Поэтому в данном случае выбрать какое-то одно значение коэффициента $\alpha$ не представляется возможным.

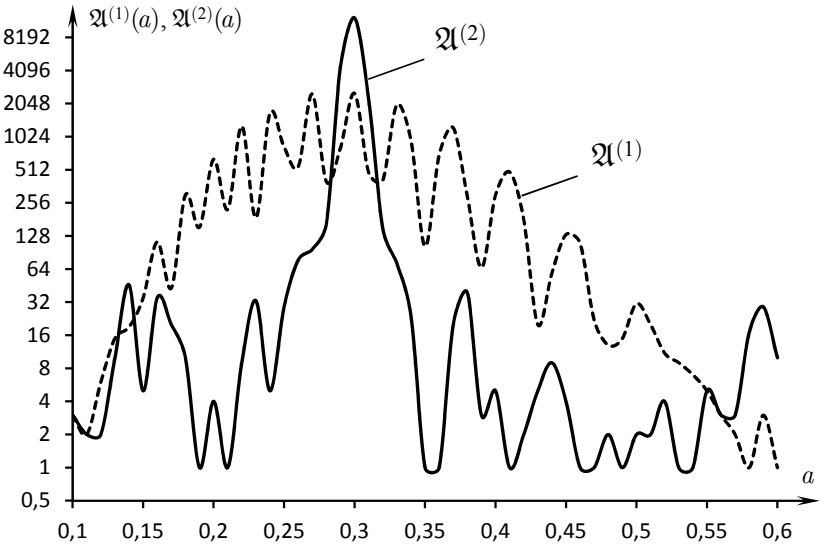



**Fig. 2.** The graphs of the functions $\mathfrak{A}^{(1)}(a)$ and $\mathfrak{A}^{(2)}(a)$

**Рис. 2.** Графики функций $\mathfrak{A}^{(1)}(a)$ и $\mathfrak{A}^{(2)}(a)$

Для каждого $k \in \mathcal{K}$ оценка качества аппроксимации функции $f_k(x)$ осуществлялась путем определения числа, ограничивающего сверху величины

$$\left|1 - g_k(x) \big/ \tilde{f}_k(x)\right| \quad \big(x \in \mathcal{P}(50000)\big)$$

по меньшей мере для 95% элементов множества $\mathcal{P}(50000)$. Будем говорить, что число $k \in \mathcal{K}$ обладает свойством $\mathfrak{P}(r)$, где $r \in (0;1)$, если

$$P\left\{\left|1 - g_k(x) \big/ \tilde{f}_k(x)\right| \le r\right\} \ge 0,95$$

или, что то же самое,

$$\operatorname{card}\left\{x \in \mathcal{P}(50000) \;\middle|\; \left|1 - g_k(x) \big/ \tilde{f}_k(x)\right| \le r\right\} \ge 47500.$$

В табл. 4 представлены сведения о количествах элементов множества $\mathcal{K}$, *не* обладающих свойством $\mathfrak{P}(r)$, в зависимости от способа вычисления коэффициента $\beta$. Как и следовало ожидать, применение первого способа дает лучшее приближение. Однако и при использовании второго способа большинство чисел из множества $\mathcal{K}$ обладает свойством $\mathfrak{P}(0,15)$, что также можно считать вполне удовлетворительным результатом.

Естественно предположить, что если для некоторого $k \in \mathcal{K}$ величина $\beta_k$ существенно отличается от $\beta_{avg}$, то функция $g_k(x) = \alpha_k^{(2)} x^{\beta_{avg}}$ не очень хорошо приближает функцию $f_k(x)$ и потому свойство $\mathfrak{P}(r)$ выполняется для числа $k$ лишь при достаточно большом $r$. Тем не менее, соответствия между величиной $|\beta_k - \beta_{avg}|$ и минимальным $r \in (0;1)$, для которого $k$ обладает свойством $\mathfrak{P}(r)$, выявлено не было.

**Table 4.** The numbers of elements of $\mathcal{K}$ that do not have the property $\mathfrak{P}(r)$

**Таблица 4.** Количества элементов $\mathcal{K}$, не обладающих свойством $\mathfrak{P}(r)$

| $r$ | 0, 08 | 0, 09 | 0, 10 | 0, 11 | 0, 12 | 0, 13 | 0, 14 | 0, 15 | 0, 16 | 0, 17 | 0, 18 | 0, 19 | 0, 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Способ 1 | 226 | 100 | 52 | 26 | 17 | 8 | 5 | 3 | 2 | 1 | 1 | 0 | 0 |
| Способ 2 | 1860 | 1091 | 670 | 383 | 233 | 133 | 79 | 55 | 35 | 24 | 14 | 5 | 4 |

Итак, проведенный анализ позволяет сделать следующий

**Вывод.** *Почти для всех* $k \in \mathcal{K}$ *функция* $f_k(x)$ *на множестве* $\{1, 2, \ldots, 50000\}$ *может быть с достаточной точностью приближена функцией* $g_k(x) = \alpha_k x^{\beta}$, *где* $\beta = \beta_{avg} = 0,85$ *и* $0,28 < \alpha_k \leq 0,31$.

## 4. О вычислительной сложности проверки условий теоремы 3

Все описанные выше результаты получены с помощью программ, реализованных автором на языке C++ (GCC) с использованием C RTL и C++ STL. Из произведенных расчетов наибольшую вычислительную сложность имело отыскание элементов множества $S_k(n)$. Для выполнения этой операции требуется проверить на соответствие условиям теоремы 3 $\frac{n(n-1)}{2}$ пар $(p, q)$, где $p < q$. В ходе каждой такой проверки необходимо убедиться в том, что

1) $p$ не делит $k - 1$;
2) $q$ не делит $k - 1$;
3) $(k, q) = 1$ (что ввиду простоты числа $q$ равносильно условию $q \nmid k$);
4) $p$ делит $q - 1$;
5) порядок $k$ по модулю $q$ является $p$-числом.

Очевидно, что первые четыре операции могут быть выполнены процессором за время $O(1)$. Если следовать математической формулировке последней операции, то необходимо

а) вычислить порядок числа $k$ по модулю $q$;
б) проверить, является ли он степенью числа $p$.

Напомним, что порядок $k$ по модулю $q$ — это наименьшее число $x$ из множества $\{0, 1, \ldots, q-1\}$, удовлетворяющее соотношению $k^x \equiv 1 \pmod{q}$. Таким образом, отыскание порядка числа $k$ — это не что иное, как вычисление дискретного логарифма в мультипликативной группе поля $\mathbb{Z}_q$. Как известно, данная задача является достаточно трудоемкой: распространенные алгоритмы ее решения имеют сложность порядка $O(\sqrt{q})$.

Чтобы сделать алгоритм более эффективным, заметим, что в действительности нам не нужно вычислять порядок $s$ числа $k$, а достаточно лишь проверить, является ли он $p$-числом. Пусть $m$ — максимальное целое число, удовлетворяющее соотношению $p^m \mid q - 1$. Так как рассматриваемая группа имеет порядок $q - 1$, то $s \mid q - 1$ и, значит, порядок $s$ является $p$-числом тогда и только тогда, когда он совпадает с одним из чисел $1, p, p^2, \ldots, p^m$. При этом $s \neq 1$, поскольку $q$ не делит $k - 1$.

Таким образом, для выполнения последней, пятой, операции необходимо $m \leq \log_p(q-1)$ проб. Каждая из них требует возвести в степень $p$ либо число $k$ (для первой пробы), либо число, рассмотренное при предыдущей пробе. При использовании бинарного алгоритма возведения в степень (см., например, [6, § 1.2]) для этого требуется порядка $O(\ell)$ умножений, где $\ell = \lfloor \log_2 p \rfloor + 1$ — длина битового представления числа $p$. В итоге, предложенный алгоритм проверки пары $(p, q)$ на соответствие условиям теоремы 3 имеет сложность

$$O(\log_2 p \cdot \log_p q) = O(\log_2 q),$$

меньшую, чем при вычислении дискретного логарифма.

## References


[1] D. I. Moldavanski and N. Y. Sibyakova, “On the finite images of some one-relator groups”, *Proc. Amer. Math. Soc.*, vol. 123, 2017–2020, 1995.

[2] G. Baumslag and D. Solitar, “Some two-generator one-relator non-Hopfian groups”, *Bull. Amer. Math. Soc.*, vol. 68, 199–201, 1962.

[3] S. Meskin, “Nonresidually finite one-relator groups”, *Trans. Amer. Math. Soc.*, vol. 164, 105–114, 1972.

[4] D. I. Moldavanskii, "The residual $p$-finiteness of HNN-extensions", *Bull. Ivanovo State Univ.*, no. 3, 129–140, 2000.

[5] O. A. Ivanova and D. I. Moldavanskii, "The residual $\pi$-finiteness of some one-relator groups", *Proc. Ivanovo State Univ. Mathematics*, vol. 6, 51–58, 2008.

[6] I. A. Pankratova, *Number-theoretic cryptography methods*. Tomsk State Univ., 2009.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 2, 2021
journal homepage: www.mais-journal.ru

COMPUTER SYSTEM ORGANIZATION

# Object-Centric Replay-Based Conformance Checking: Unveiling Desire Lines and Local Deviations

J. C. Carrasquel[1], K. Mecheraoui[2]

DOI: 10.18255/1818-1015-2021-2-146-168

[1]HSE University, 20 Myasnitskaya str., Moscow 101000, Russia.
[2]University of Constantine 2 – Abdelhamid Mehri, Nouvelle ville Ali Mendjeli BP : 67A, 25000 Constantine, Algeria.

MSC2020: 68N30
Research article
Full text in English

Received May 3, 2021
After revision May 25, 2021
Accepted June 2, 2021

Conformance checking methods diagnose to which extent a real system, whose behavior is recorded in an event log, complies with its specification model, e.g., a Petri net. Nonetheless, the majority of these methods focus on checking isolated process instances, neglecting interaction between instances in a system. Addressing this limitation, a series of object-centric approaches have been proposed in the field of process mining. These approaches are based on the holistic analysis of the multiple process instances interacting in a system, where each instance is centered on the handling of an object. Inspired by the object-centric paradigm, this paper presents a replay-based conformance checking method which uses a class of colored Petri nets (CPNs) – a Petri net extension where tokens in the model carry values of some types (colors). Particularly, we consider conservative workflow CPNs which allow to describe the expected behavior of a system whose components are centered on the end-to-end processing of distinguishable objects. For describing a system's real behavior, we consider event logs whose events have sets of objects involved in the execution of activities. For replay, we consider a jump strategy where tokens absent from input places of a transition to fire move from their current place of the model to the requested places. Token jumps allow to identify desire lines, i.e., object paths unforeseen in the specification. Also, we introduce local diagnostics based on the proportion of jumps in specific model components. The metrics allow to inform the severity of deviations in precise system parts. Finally, we report experiments supported by a prototype of our method. To show the practical value of our method, we employ a case study on trading systems, where orders from users are matched to trade.

**Keywords:** Process Mining; Conformance Checking; Petri Nets; Colored Petri Nets

INFORMATION ABOUT THE AUTHORS

Julio C Carrasquel
correspondence author
orcid.org/0000-0003-3557-797X. E-mail: jcarrasquel@hse.ru
Postgraduate Student.

Khalil Mecheraoui
orcid.org/0000-0001-9906-6074. E-mail: k_mecheraoui@esi.dz
Assistant Professor.

**Funding:** This work is supported by the Basic Research Program at the National Research University Higher School of Economics.

**For citation**: J. C. Carrasquel and K. Mecheraoui, "Object-Centric Replay-Based Conformance Checking: Unveiling Desire Lines and Local Deviations", *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 146-168, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 2, 2021
сайт журнала: www.mais-journal.ru

COMPUTER SYSTEM ORGANIZATION

# Объектно-ориентированная проверка соответствия модели на основе воспроизведения журнала событий: выявление желаемого поведения и локальных отклонений

Х. С. Карраскель[1], Х. Мешерауи[2]

DOI: 10.18255/1818-1015-2021-2-146-168

[1]Национальный исследовательский университет "Высшая школа экономики", ул. Мясницкая, д. 20, г. Москва, 101000 Россия.
[2]Университет Константина 2 - Абделхамид Мехри, Нувель виль Али Менджели ВР : 67А, 25000 Константин, Алжир.

УДК 519.686.2
Научная статья
Полный текст на английском языке

Получена 3 мая 2021 г.
После доработки 25 мая 2021 г.
Принята к публикации 2 июня 2021 г.

Методы проверки соответствия позволяют установить, в какой степени реальная система, поведение которой регистрируется в журнале событий, соответствует ее модели, например, в виде сети Петри. Большинство таких методов направлены на проверку изолированных экземпляров процесса и игнорируют взаимодействие между экземплярами в системе. Для преодоления этого ограничения в области интеллектуального анализа данных был предложен ряд объектно-ориентированных подходов. Эти подходы основаны на целостном анализе нескольких экземпляров процесса, взаимодействующих в системе, где каждый экземпляр соответствует некоторому объекту. В этой статье объектно-ориентированный подход применяется к методу проверки соответствия между журналами событий и цветными сетями Петри (CPN) – расширением сетей Петри, в котором фишки в модели представляют собой значения некоторых типов (цветов). В частности, мы рассматриваем консервативные CPN потоков работ, которые позволяют описывать ожидаемое поведение системы, в которой компоненты соответствуют обработке различных объектов. Реальное поведение системы описано в журнале событий, в котором события атрибутированы участвующими в них объектами. Для воспроизведения журнала событий мы используем стратегию прыжков, когда фишки, необходимые для срабатывания перехода, перемещаются из своих текущих позиций во входные позиции этого перехода. Прыжки фишек позволяют идентифицировать линии желаний, то есть поведения объектов, не предусмотренные в спецификации. Также мы представляем локальную диагностику, основанную на доле прыжков в поведении конкретных компонент модели. Эти метрики позволяют судить о серьезности отклонений в тех или иных частях системы. Наконец, мы приводим эксперименты, выполненные с помощью программного прототипа. Практическая ценность нашего метода показана на примере моделирования торговых систем, при котором устанавливаются соответствия между заявками пользователей и сделками.

**Ключевые слова:** Process Mining; проверка соответствия; сети Петри; Раскрашенные сети Петри

## ИНФОРМАЦИЯ ОБ АВТОРАХ

Хулио С Карраскель
автор для корреспонденции
orcid.org/0000-0003-3557-797X. E-mail: jcarrasquel@hse.ru
аспирант.

Халил Мешерауи
orcid.org/0000-0001-9906-6074. E-mail: k_mecheraoui@esi.dz
преподаватель.

**Финансирование:** Исследование выполнено при поддержке Программы фундаментальных исследований НИУ ВШЭ.

**Для цитирования**: J. C. Carrasquel and K. Mecheraoui, "Object-Centric Replay-Based Conformance Checking: Unveiling Desire Lines and Local Deviations", *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 146-168, 2021.

## 1. Introduction

Process mining is a discipline that focuses on the analysis of system processes on the basis of event logs and formal models [1]. Event logs consist of recorded traces, which describe the *real behavior* of systems. As formal models, most process mining methods use Petri nets — a formalism for analysis of concurrent distributed systems [2]. Thus, process mining methods allow, for instance, to discover models describing the real processes from event logs, or to check compliance of real processes by comparing their logs with models describing *expected behavior*. The former kind of method is called *process discovery*, whereas the latter is referred to as *conformance checking* [3]. These methods have increasingly gained attention, resulting in a plethora of case studies from business organizations [4] and innovative research applications, e.g., see [5–7].

Nevertheless, the majority of process mining methods have hitherto consisted on the individual analysis of *isolated* process instances, thereby neglecting their interaction with other instances in the system. This assumption falls short, and may throw out a wrong analysis, especially when there is a strong dependency between the life-cycles of instances (for example, "a buy order is filled only if a sell order is in the system, and both orders are in certain locations").

To overcome such limitations, *object-centric* approaches have emerged as a popular paradigm in the field of process mining [8, 9]. The common denominator of these approaches lies in the holistic analysis of the multiple processes interacting in a system, where each process is centered on the management of a class of objects. Thus, within this research direction, novel multi-instance modeling notations [10, 11], process discovery methods [12], and formal verification techniques [13] have been proposed.

As for conformance checking, certain methods have been proposed to validate multiple perspectives of a process beyond the control-flow, i.e., the correct ordering of activities [14, 15]. Particularly, these methods make use of Petri nets with data (DPNs) to detect deviations caused by data corruptions (e.g., "a loan approval was wrongly executed due to a requested amount higher than expected"). However, the backbone of DPNs is an ordinary Petri net used to describe the individual execution of a single process instance, and data elements are only statically attached to model transitions. Thus, this model does not allow to describe and validate the dynamic and concurrent interaction of multiple instances within a system.

In this paper, inspired by the object-centric paradigm, we present a conformance checking method to diagnose whether systems that manage different kinds of objects comply with their specification. The method uses a class of colored Petri nets (CPNs) — a Petri net extension where tokens carry values representing objects of different types (called *colors*) [16]. Particularly, we consider *conservative workflow* CPNs with multiple source and sink places. In this model, tokens cannot disappear or duplicate, and they move through paths whose endpoints are specific pairs of source and sink places. In this way, the model allows to describe the expected behavior of systems with components centered on the end-to-end processing of distinct objects of a certain type. For describing the system's real behavior, we consider traces of event logs, where events consist of executed activities and sets of object identifiers. The latter indicates which objects were involved in an event's activity. As we will present, the characteristics proposed for both CPNs and event logs allow us not only to keep track of individual objects, but also to provide an algorithm with linear time complexity.

Our method is based on replaying individually each trace of an event log on top of a CPN model. When replaying each trace, the distinct objects are injected as tokens in source places of the model. Then, for each event of the trace, we seek to fire a transition labeled with the event's activity, and selecting tokens from the transition's input places that correspond to the event's object identifiers. If a token related to an event's object is not in a requested input place, we consider a *jump strategy*, where the missing token is moved from its current location in the model to the requested input place. This allows to force a transition firing, and to keep replaying a trace to find more deviations. The method reports all token jumps between places and their frequency. As we will present, this information can be added to the input CPN model to unveil so-called *desire lines* [17], i.e., actual paths of objects which are unforeseen in the specification model.

In addition, we present *local conformance metrics* based on the proportions of token transfers and jumps through specific components of a CPN model. By leveraging the correspondence of parts of a real system with components of a CPN model (e.g., activities with transitions and system locations with places), these metrics then allow to diagnose local deviations and their severity in concrete parts of a system. Finally, we report an experimental evaluation supported by a prototypical implementation of our method. To showcase the practical value of our approach, we shall make use of a case study on trading systems, where orders from users are matched to trade.

The remainder of this paper is structured as follows. Section 2 introduces a motivating example illustrating the use of our method in trading systems. Sections 3 and 4 present the class of CPNs and event logs, which we use in our method. Section 5 presents the conformance checking method, as well as the conformance metrics. Section 6 reports experimental evaluations supported by our prototype. Section 7 discusses the related work, and finally Section 8 presents the conclusions.

## 2. Motivating Example: Checking Conformance in Trading Systems

To give the reader an idea on how our method can be applied on a specific domain, let us consider the validation of *trading systems* in stock exchanges [18]. In trading systems, buy orders and sell orders from users are crossed to trade securities (e.g., stocks of a company). Orders that aim to trade the same kind of securities are placed in two-sided lists called *order books* (e.g., orders that buy or sell securities of "`rosneft`" are placed in the order book "`rosneft`"). In a trading system, there are as many order books as kinds of securities that can be traded.

Fig. 1 presents a CPN modeling the specification of a trading system operating one order book. It describes how the system is expected to manage both object classes, buy orders and sell orders. CPNs consist of two kinds of nodes: places and transitions. Places (drawn as circles) represent locations. For example, places $p_1$ and $p_2$ are source places for orders, $p_3$ and $p_4$ model buy and sell sides of the order book, and finally $p_5$ and $p_6$ are sink places for canceled or filled orders. Conversely, transitions (drawn as boxes) model system activities. Transitions consume tokens from input places, producing them back in output places. Thus, transitions $a$ and $b$ model submission of orders, transitions $c$ and $d$ represent cancellation of orders, whereas transition $e$ models a trade execution between two orders. Albeit this example abstracts from other activities in a trading system, it will allow us to clearly illustrate our method.

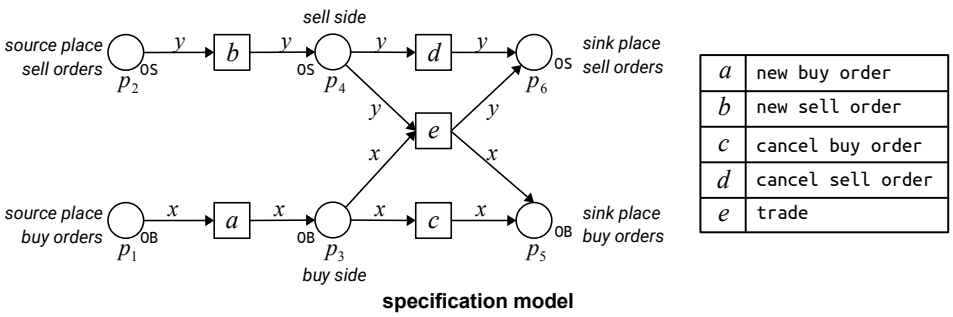
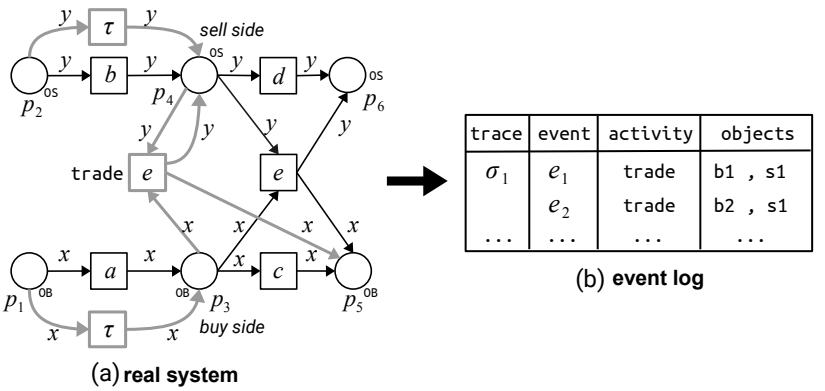



Fig. 1: CPN model specifying the *expected behavior* of a trading system operating an order book

Like other distributed technologies, trading systems are prone to failures, e.g., due to errors during the system's development or due to malicious users hacking the system. Thus, trading systems are sensitive to deviate from their specification. For instance, let us assume a trading system initially built upon the specification of Fig. 1, but whose real behavior is actually described in Fig. 2(a). In this real system, buy and sell orders are "silently" allowed to trade without being accepted in the order book, i.e., skipping activities $a$ and $b$. Also, an undesired variant of activity $e$ allows sell orders to trade an unrestricted number of times.

For system engineers, it can be hard to determine these deviations and to which extent they have violated the system. Fortunately, event logs of this system record this misbehavior (e.g., Fig. 2(b)). For instance, the two first events in the trace $\sigma_1$ of Fig. 2(b) exemplify the mentioned problems of this system: a buy order `b1` and a sell order `s1` have traded, skipping activities *a* and *b*. Also, the sell order `s1` illegally trades in the next event. Powered by this kind of logs, we introduce how our conformance method unveils these deviations.




| trace | event | activity | objects |
|---|---|---|---|
| $\sigma_1$ | $e_1$ | trade | b1 , s1 |
| | $e_2$ | trade | b2 , s1 |
| ... | ... | ... | ... |

Fig. 2: CPN describing the *real behavior* of a trading system with undesired behavior highlighted in gray (a). Our method takes an *event log* (b) from this real system.

Fig. 3 illustrates the setting of our method for detecting the aforementioned deviations. As input, the method takes the CPN specification model of Fig. 1, and an event log of the real system as exemplified in Fig. 2. In this example, each trace of an event log corresponds to all events executed in a specific order book. As output, the method produces a report listing all token jumps detected (representing objects skipping activities), global and local conformance metrics, as well as other traffic statistics. These reports can be used, for example, to extend the specification model in order to clearly identify deviations and their magnitude.

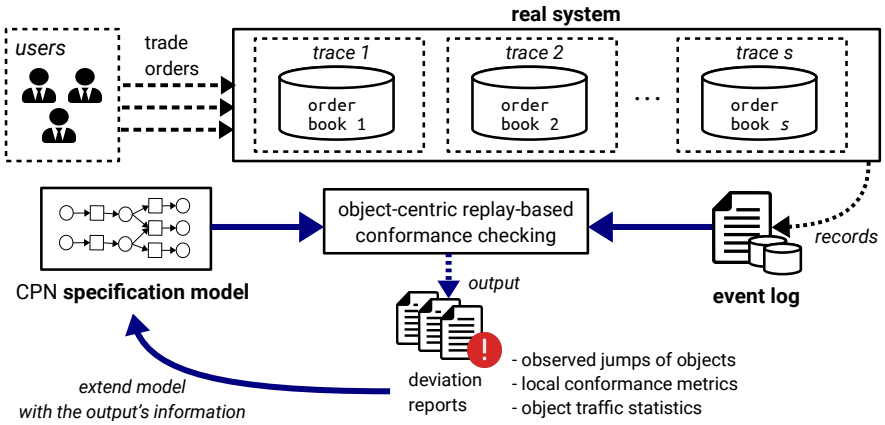



Fig. 3: Validating real behavior of a trading system via object-centric replay-based conformance.

Fig. 4 shows an example of the specification model extended with the mentioned outputs of our method. Token jumps and their average frequencies on traces appear as dotted lines between places. Note how these jumps directly correlate with the components in Fig. 2(a) that describe undesired behavior of the real system, i.e., orders skipping activities *a* and *b* jump from places $p_1$, $p_2$ to places $p_3$ and $p_4$. Also, components of the extended CPN are labeled with traffic statistics. For instance, places indicate how many tokens were transferred from them (`k`), and how many of them actually arrived to the place by a jump (`j`).

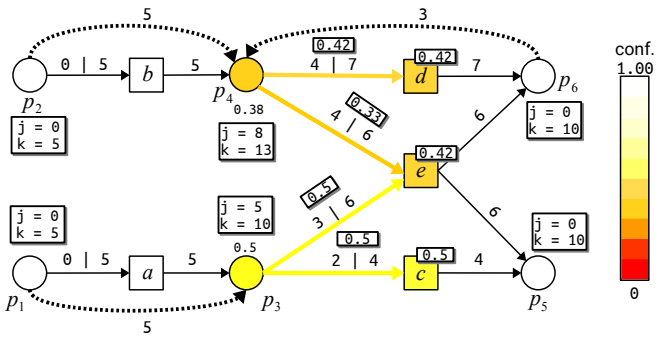



Fig. 4: Specification model of Fig. 1 enriched with the output information of our conformance method.

By leveraging the association between model's transitions and places with system's activities and locations, local conformance metrics allow to diagnose the severity of deviations on precise system components. For instance, one of these metrics checks the proportion of token jumps to input places of a concrete transition to force its firing. Then, this allows to measure the proportion of objects executing a precise activity without following the specification path. For example, activity $d$ consumed 7 orders, but 4 of them were not ready at the location related to place $p_4$. Thus, this activity is associated with a measure of 0.42, i.e., 42% of the objects were at the required location when executing $d$, whereas the rest were improperly involved in this activity. As we will present in the next sections, similar local metrics can be associated to places (locations) and arcs from places to transitions. Finally, as depicted in Fig. 4, the metrics can be combined with the notion of a *heat map* to clearly visualize which components of a system have been violated more, e.g., if the local measure of a component is close to 0, then such a component is painted in red.

## 3. Colored Petri Nets

In this section, we present formal definitions and execution semantics for a class of colored Petri nets (CPNs). As introduced in Section 1, CPNs are an extension of Petri nets where tokens carry values of some types (also referred to as *colors*). For example, types may account for object classes, whereas tokens carry object identifiers. Let $\mathfrak{D} = \{D_1, ..., D_k\}$ be a finite set of types. Each token in a CPN carries a value $\mathtt{d}$ of some type $D \in \mathfrak{D}$. For instance, in the model of Fig. 1 two types are defined: `OB` for buy orders and `OS` for sell orders. Places are mapped to types in $\mathfrak{D}$ to indicate the kind of tokens they contain, e.g., $\mathtt{type}(p_1) = \mathtt{OB}$.

Arcs are labeled with expressions to describe how tokens are processed upon transition firings. We consider that each expression consists of a typed variable. Let $\mathcal{V}$ be a finite set of typed variables. We denote by $\mathtt{type}(\mathtt{v})$ the type of a variable $\mathtt{v} \in \mathcal{V}$ s.t. $\mathtt{type}(\mathtt{v}) \in \mathfrak{D}$. We define a function $W$ that maps each arc to a variable in $\mathcal{V}$. For example, in Fig. 1, expressions $W(p_1, t_1) = W(t_1, p_3) = \mathtt{x}$ specify that one buy order is transferred from place $p_1$ to $p_3$ upon the firing of $t_1$. Let $\mathcal{A}$ be a finite set of *activities*. To relate transitions with real system activities, we fix an *activity-labeling function* $\Lambda$ that maps transitions to elements in $\mathcal{A}$.

**Definition 1 (Colored Petri Net).** *Let $\mathfrak{D}$ be a finite set of types, let $\mathcal{V}$ be a finite set of variables typed over $\mathfrak{D}$, and let $\mathcal{A}$ be a finite set of activities. A colored Petri net is a 6-tuple $CP = (P, T, F, \mathtt{type}, W, \Lambda)$ where:*

- *$P$ is a finite set of places;*
- *$T$ is a finite set of transitions, s.t. $P \cap T = \emptyset$;*
- *$F \subseteq (P \times T) \cup (T \times P)$ is a finite set of directed arcs (called the flow relation);*
- *$\mathtt{type} : P \longrightarrow \mathfrak{D}$ is a place-typing function, mapping each place to a type in $\mathfrak{D}$;*
- *$W : F \longrightarrow \mathcal{V}$ is an arc-labeling function, mapping each arc $r$ to a variable in $\mathcal{V}$. $\forall r \in F$, if $r$ is adjacent to a place $p \in P$, then $\mathtt{type}(W(r)) = \mathtt{type}(p)$;*
- *$\Lambda : T \longrightarrow \mathcal{A}$ is an activity-labeling function, s.t. $\forall t, t' \in T : t \neq t' \iff \Lambda(t) \neq \Lambda(t')$, i.e., transitions are mapped to distinct activities.*

We now define execution semantics for the CPNs defined above. Let $CP = (P, T, F, \texttt{type}, W, \Lambda)$ be a CPN. A *marking* $M$ in a CPN is a function, mapping every place $p \in P$ to a (possibly empty) set of tokens $M(p)$, such that $M(p) \subseteq \texttt{type}(p)$. We denote by $M_0$ an initial marking. A *binding* $b$ of a transition $t \in T$ is a function, that assigns a value $b(\texttt{v})$ to each variable $\texttt{v}$ occurring in arc expressions adjacent to $t$, such that $b(\texttt{v}) \in \texttt{type}(\texttt{v})$. Let ${}^\bullet t$ and $t^\bullet$ be respectively the sets of input places and output places of a transition $t \in T$. Transition $t$ is *enabled* in marking $M$ w.r.t. a binding $b$ iff $\forall p \in {}^\bullet t \ : \ b(W(p,t)) \in M(p)$, that is, each input place of $t$ has at least one token to be consumed. The *firing* of an enabled transition $t$ in a marking $M$ w.r.t. to a binding $b$ yields a new marking $M'$ such that $\forall p \in P \ : \ M'(p) = M(p) \setminus \{b(W(p,t))\} \cup \{b(W(t,p))\}$.

As introduced in Section 1, we make use of CPNs to model systems consisting of components centered on the *end-to-end processing* of different types of *distinguishable objects*. To faithfully describe these systems, CPNs shall be characterized by the following properties: on the one hand, CPNs must be *conservative*, that is, tokens cannot disappear or duplicate upon transition firings; on the other hand, models must be *workflow-oriented* with a pair of source and sink places for every type defined in the model, such that tokens move in a path between a source and a sink corresponding to their type. We thus define *conservative workflow* CPNs.

**Definition 2 (Conservative-Workflow Colored Petri Net).** *Let $\mathfrak{D} = \{D_1, ..., D_k\}$ be a finite set of types such that $k \geq 1$, and let $CP = (P, T, F, \texttt{type}, W, \Lambda)$ be a CPN defined over $\mathfrak{D}$. We say that $CP$ is a conservative-workflow CPN if and only if:*

1. *$CP$ is a conservative colored Petri net where:*
   - $\forall\ t \in T\ \ \forall\ p \in {}^\bullet t\ \ \exists!\ p' \in t^\bullet \ : \ W(p,t) = W(t,p')$.
   - $\forall\ t \in T\ \ \forall\ p \in t^\bullet\ \ \exists!\ p' \in {}^\bullet t \ : \ W(p',t) = W(t,p)$.
2. *For every $j \in \{1, ..., k\}$, there exists one distinguished pair of places in $P$, a source place $i_j$ and a sink place $o_j$ in $P$, where $\texttt{type}(i_j) = \texttt{type}(o_j) = D_j$ with $D_j \in \mathfrak{D}$, and there exists a path in $CP$ from $i_j$ to $o_j$ such that for every place $p$ in the path $\texttt{type}(p) = D_j$. We denote by $P_0$ and $P_F$ the sets of source places and sink places in $CP$.*
3. *$\forall t \in T : \ \forall p, p' \in {}^\bullet t \ p \neq p' \iff \texttt{type}(p) \neq \texttt{type}(p') \ \wedge \ \forall p, p' \in t^\bullet \ p \neq p' \iff \texttt{type}(p) \neq \texttt{type}(p')$, i.e., for every transition $t$, places located within the set of input places of $t$ have distinct types. The same rule holds for places located in the set of output places of $t$.*

As input models for our method, we shall consider conservative workflow CPNs. We briefly explain Definition 2. Firstly, a CPN is conservative iff for every variable $\texttt{v}$ occurring in an input arc of a transition $t$, $\texttt{v}$ occurs exactly once in an output arc of $t$. Similarly, each variable occurring in an output arc of $t$ shall occur exactly once in an input arc of $t$. This implies that when token values in input places are binded to variables upon transition firings, then such values are transferred to output places, without disappearing or being duplicated. In this way, our conformance method will be able to unambiguously associate every distinct object in a trace with a token in the model.

Also, according to Definition 2, CPNs shall be workflow-oriented. More precisely, a workflow CPN has distinct $k$ source places and $k$ sink places where $k$ is the number of types defined in the model. Each pair of source and sink places of the same type are connected by a path whose intermediate places are also of the same type. In our conformance method, distinct objects in a trace are injected as tokens in source places. Then, tokens move in paths according to the information of objects recorded in events. Upon termination, the method will check whether these tokens arrived to their corresponding sink places. It can be inferred that all places of the same type form a subnet within a workflow CPN, where each sub-net represents a system component handling end-to-end processing of a concrete object class.

Finally, Definition 2 states that the model does not have transitions with input places of the same type. The latter allows to relate every object type with exactly one input place in each transition. In Section 5, we discuss how this syntactic restriction contributes to providing an algorithm with linear time complexity.

## 4. Event Logs

In this section, we introduce *event logs*, describing how they are structured.

**Definition 3 (Event Log, Trace, Event).** *An event log is a finite set of traces $L = \{\sigma_1, ..., \sigma_s\}$ where, for each $i \in \{1, ..., s\}$, a trace $\sigma_i = \langle e_1, ..., e_m \rangle$ is a finite sequence of events, s.t. $m = |\sigma_i|$ is the trace length.*

*Let $\mathcal{A}$ be a finite set of activities, and let $\mathfrak{D}$ be a finite set of types. For every trace $\sigma$ in $L$, each event $e$ in $\sigma$ is a tuple of the form $e = (a, R(e))$, where $a \in \mathcal{A}$ is an activity label, and $R(e) = \{r_1, ..., r_k\}$ is a finite set of objects. For each $j \in \{1, ..., k\}$, we say that $r_j \in R(e)$ is an object of type $D$ involved in the execution of activity $a$, such that $D \in \mathfrak{D}$.*

Table 1: An event log $L$ with two examples of traces, which correspond to runs in a trading system.

| **trace** | **event** ($e$) | **activity** ($a$) | **objects** ($R(e)$) |
|---|---|---|---|
| $\sigma_1$ | $e_1$ | `new buy order` | `b1` |
| | $e_2$ | `new sell order` | `s1` |
| | $e_3$ | `new sell order` | `s2` |
| | $e_4$ | `trade` | `b1 , s1` |
| | $e_5$ | `cancel sell order` | `s2` |
| $\sigma_2$ | $e_1$ | `new buy order` | `b1` |
| | $e_2$ | `trade` | `b1 , s1` |
| | $e_3$ | `trade` | `b2 , s1` |
| | $e_4$ | `new sell order` | `s2` |

Table 1 presents an event log with two traces related to end-to-end runs in a trading system. Events indicate activities executed and objects involved, e.g., event $e_4$ in trace $\sigma_1$ indicates an execution of activity `trade` with two objects involved: buy order `b1` and sell order `s1`. We consider that all objects in a trace are distinguishable by having distinct identifiers. We denote by $R(\sigma)$ the set of distinct objects in a trace, e.g., $R(\sigma_1) = \{\texttt{b1}, \texttt{s1}, \texttt{s2}\}$. With slight abuse of notation, we denote the type of an object $r$ by $\texttt{type}(r)$.

To guarantee the proper execution of our method, event logs must comply with a criterion of *syntactical correctness* with respect to the CPN used in the method. Let $L$ be an event log, and let $CP = (P, T, F, \texttt{type}, W, \Lambda)$ be a conservative workflow CPN. We say that $L$ is *syntactically correct* w.r.t. to $CP$ iff, for every trace $\sigma \in L$, each event $e$ in $\sigma$ is syntactically correct. An event $e = (a, R(e))$ is syntactically correct w.r.t. to $CP$ iff $\exists t \in T : \Lambda(t) = a \wedge \forall p \in {}^\bullet t \ \exists! r \in R(e) : \texttt{type}(r) = \texttt{type}(p) \wedge \forall r \in R(e) \ \exists! p \in {}^\bullet t : \texttt{type}(r) = \texttt{type}(p)$. That is, for every event $e = (a, R(e))$, there exists a transition $t$ labeled with activity $a$, and each input place of $t$ is associated with exactly one event's object, and similarly each event's object is associated with exactly one input place of $t$.

## 5. Object-Centric Replay-Based Conformance Checking

### 5.1. The Algorithm

In this section, we present our conformance checking method. The method is based on the replay strategy described in [1] with some adaptations for the class of CPNs and event logs described in Sections 3 and 4. Particularly, we shall assume that input models are conservative workflow CPNs (i.e., see Definition 2), whereas event logs are syntactically correct to these models.

The method replays individually each trace of an event log on a CPN. We consider that the input CPN has an empty initial marking. When replaying a trace $\sigma$, distinct objects in $\sigma$ are firstly inserted as tokens in source places of the CPN, according to their type. Then, for each event $e = (a, R(e))$ in $\sigma$, the method seeks to fire a transition $t$ labeled with activity $a$, and consuming the tokens that correspond to the event's objects, i.e., elements in $R(e)$. If a token corresponding to an event's object is not in an input place of $t$, then we consider a *jump strategy* where the missing token is moved from their current location in the model to

the requested input place. This allows to force transition firings, and to keep replaying a trace to find more deviations.

Algorithm 1 describes the method. As output, the method returns two integer counters `j` and `k`. The value `j` is the total number of token jumps, whereas the value `k` is the total number of tokens transferred from input places to output places upon transition firings. A ratio between these values `j` and `k` allows to measure the discrepancy between a trace and a CPN. In the following, we illustrate the use of the algorithm, whereas at the end of this part we introduce conformance measures based on these counters.

**Algorithm 1: Replay on CPNs with a token jump strategy**

**Input:** $CP = (P, T, F, \texttt{type}, W, \lambda)$ — conservative-workflow CPN with marking $M$ initially empty;
$P_0, P_F \subseteq P$ — non-empty sets of source and sink places;
$\sigma$ — an event log trace;

**Output:** `j` — number of token jumps;
`k` — number of tokens consumed/produced;

```
1  j ← 0; k ← 0;
2  populateSourcePlaces(P_0, R(σ));
3  foreach e = (a, R(e)) in σ do
4      t ← selectTransition(a);
5      foreach r in R(e) do
6          if r ∉ M(p) such that p ∈ •t ∧ type(p) = type(r) then        // location[r] ≠ p
7              jump(r, p);
8              j ← j + 1;
9          endif
10     endfor
11     fire(t, R(e));
12     k ← k + |R(e)|;
13 endfor
14 foreach r in R(σ) do
15   if r ∉ M(p) such that p ∈ P_F ∧ type(p) = type(r) then
16     jump(r, p);
17     j ← j + 1;
18   endif
19 endfor
20 consumeAllObjectsFromSinkPlaces(P_F, R(σ));
21 k ← k + |R(σ)|;      // count final transfers:  consumption of all distinct objects from the sinks
22 return (j, k);
```

To illustrate how the algorithm works, we will consider the example depicted in Fig. 5, which describes step-by-step the replay of trace $\sigma_2$ in Table 1 on the CPN of Fig. 1. For compactness, transition names are used instead of activity labels. Firstly, our method extracts all distinct objects of the trace, inserting them in source places according to their type (function `populateSourcePlaces`). For example, four objects are extracted from $\sigma_2$ — buy orders `b1`, `b2` and sell orders `s1`, `s2`. Thus, the source place $p_1$ for buy orders ($\texttt{type}(p_1) = \texttt{OB}$) is populated with tokens `b1` and `b2`, and the source place $p_2$ for sell orders ($\texttt{type}(p_2) = \texttt{OS}$) is populated with tokens `s1` and `s2`.

After populating source places with the distinct objects, we start to replay the trace on the CPN. As described in lines 3-13 of Algorithm 1, for each event $e = (a, R(e))$, the following steps are performed. We select a transition $t$ in the CPN labeled with activity $a$. Then, for every object $r \in R(e)$, we check if the input place $p$ of $t$ contains object $r$. If the latter is not true, then $r$ is moved from its current location in the model to place $p$. Each token jump is counted by incrementing the value of counter `j`. Afterwards, when all observed objects in $R(e)$ are located in the input places of transition $t$, then $t$ fires. The transition consumes such objects from input places, transferring them to its output places. The counter `k` is incremented by the number of tokens transferred.

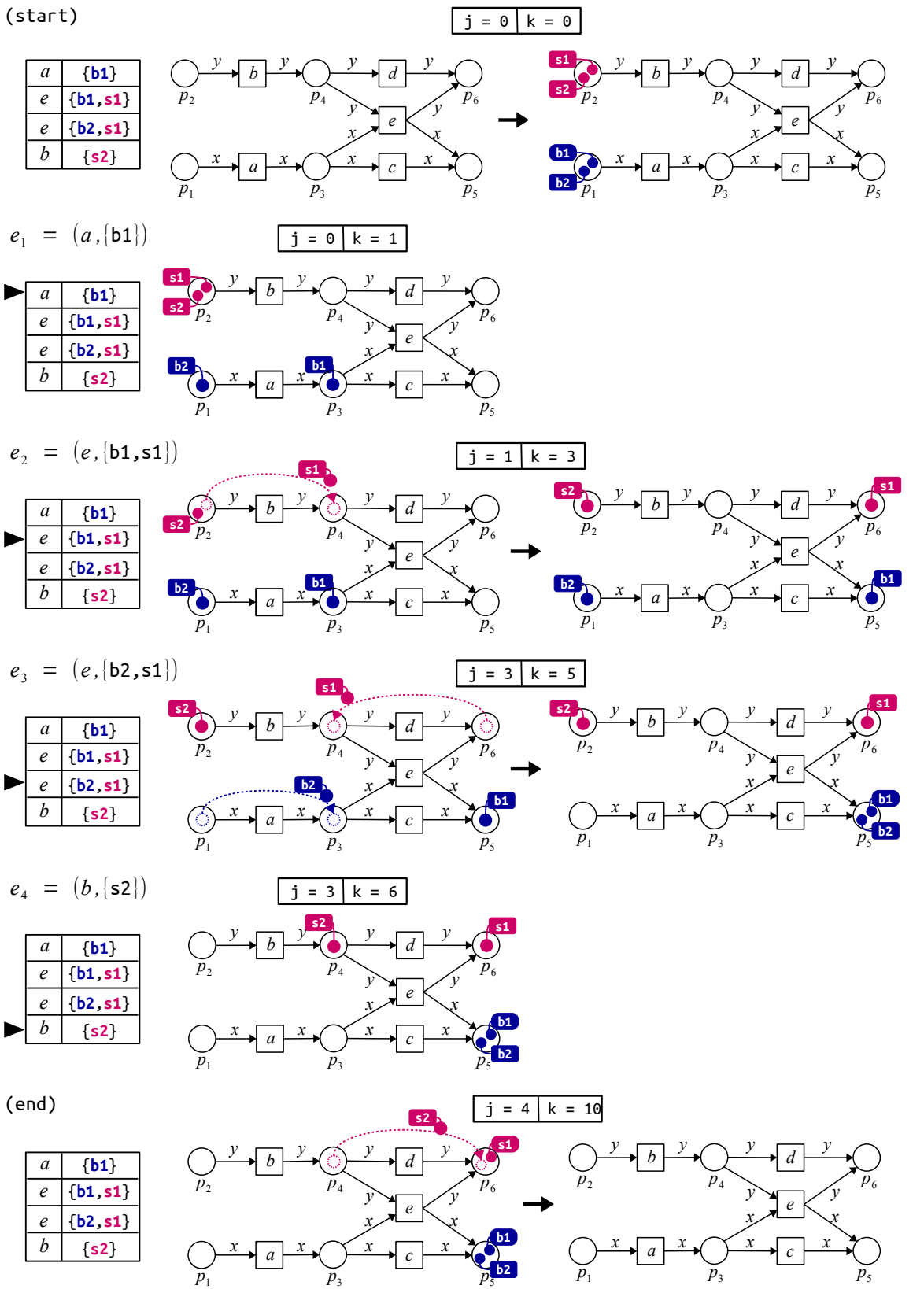



Fig. 5: Replay of trace $\sigma_2$ of Table 1 on top of the CPN of Fig. 1 using Algorithm 1.

As an example, let us focus on the replay of $e_2 = (e, \{\texttt{b1}, \texttt{s1}\})$ depicted in Fig. 5. The sell order `s1` is not in place $p_4$. This event corresponds to the situation of a trade between `b1` and `s1`, but `s1` was still not allowed to trade. However, to continue to replay, the object `s1` jumps from its current location (place $p_2$) to place $p_4$. As observed later in Fig. 5, the same situation occurs on $e_3 = (e, \{\texttt{b2}, \texttt{s1}\})$, where both tokens are absent from the input places of the transition to fire. Thus, after forcing the replay of $e_2$ note how we can detect another deviating events.

After replaying all events in a trace, we check if all distinct objects arrived to their corresponding sink places. This final step allows to validate, for example, if the real system completely processed all objects. Lines 14-21 of Algorithm 1 describe this final step. For instance, in Fig. 5, object `s2` was left in an intermediate place. This can be interpreted as a corrupt order that should have traded or been canceled at the end of a day. Hence, we force this token to jump to its sink place, which is $p_6$ since $\texttt{s2} \in \texttt{OS}$ and $\texttt{type}(p_6) = \texttt{OS}$. When all tokens are in the sink places of the CPN, they are consumed by the "environment". Note that the counter of transfers `k` is incremented by the number of all distinct objects consumed in this final step.

**Time Complexity.** We briefly analyze the time complexity of our method. Let $n = (\sum_{\forall e \in \sigma} |R(e)|)$ be the number of objects recorded in all events of a trace $\sigma$. Let $T(n)$ be the time taken to execute Algorithm 1. We sketch out that $T(n)$ is $O(n)$, where $O(n)$ is the standard asymptotic notation, referring that the execution time of the method *linearly growths* according to the number of objects $n$ in all events of a trace. This bound can be guaranteed under the assumption that access to elements of a CPN model only requires constant time, i.e., the time taken to visit a transition or a place given its name is negligible.

Let us define $T(n) = T_{\texttt{init}}(n) + T_{\texttt{rep}}(n) + T_{\texttt{end}}(n)$, where $T_{\texttt{init}}(n)$ is the execution time of the function `populateSourcePlaces` (line 2 in Algorithm 1), $T_{\texttt{rep}}(n)$ is the time taken to replay all events in trace $\sigma$ (lines 3-13), and finally $T_{\texttt{end}}(n)$ is the time taken to consume all tokens from sink places (lines 14-20). First, the function `populateSourcePlaces` visits all objects in all events of a trace $\sigma$, looking for the set of all distinct objects $R(\sigma)$, so this operation takes up to $n$ steps. Then, each distinct object $r \in R(\sigma)$ is inserted in the source place of type $\texttt{type}(r)$, which can take up to $|R(\sigma)| \leq n$ steps. Thus, $T_{\texttt{init}}(n)$ is $O(n)$.

Now, we sketch that the time taken $T_{\texttt{rep}}(n)$ to replay all events is $O(n)$. This part of the algorithm performs $n$ iterations as it visits each object $r \in R(e)$ of every event $e$ in $\sigma$. Below, we rewrite Lines 3-13 of Algorithm 1 illustrating that the operations performed for every object can be performed in constant time.

```
1  foreach e = (a, R(e)) in σ do
2    foreach r in R(e) do
3      p ← inputPlace[a, type(r)];
4      if location[r] ≠ p then                    // r ∉ M(p) such that p ∈ •t ∧ type(p) = type(r)
5        marking[location[r]].remove(r);
6        marking[p].insert(r);
7        location[r] ← p;                                                    // jump(r, p)
8        j ← j + 1;
9      endif
10     marking[p].remove(r);
11     marking[outputPlace[a, type(r)]].insert(r);  // part of firing:  r transferred to an output place
12     location[r] ← outputPlace[a, type(r)];
13     k ← k + 1;
14   endfor
15 endfor
```

In the code above, we consider that the CPN is stored in the following associative arrays: `location` tracks the position of each object in the CPN; `marking` stores the tokens contained by every place, and `inputPlace` and `outputPlace` indicates input/output places of a transition given a type. Notably, as shown above in Line 3, each object is directly related by its type to exactly one input place as we consider CPNs whose transitions have input places of distinct types (i.e., Definition 2) and events are syntactically correct w.r.t. to the CPN.

Then, if the associative arrays representing the CPN guarantee constant time to access, remove and insert elements, then it follows that the operations for every object in each event are performed in constant time. Thus, the execution time of the trace replay only depends on the number of objects $n$ in the trace, following that $T_{\texttt{rep}}(n)$ is $O(n)$. The time taken $T_{\texttt{end}}(n)$ in Lines 14-20 of Algorithm 1 is also $O(n)$ as $R(\sigma) \leq n$ distinct objects are consumed from sink places of the model. Finally, since $T_{\texttt{init}}(n)$, $T_{\texttt{rep}}(n)$, and $T_{\texttt{end}}(n)$ are $O(n)$, then the execution time $T(n) = T_{\texttt{init}}(n) + T_{\texttt{rep}}(n) + T_{\texttt{end}}(n)$ of Algorithm 1 is also $O(n)$.

**Fitness Metric.** We introduce a global metric, namely fitness, to measure the overall degree of conformance between a trace of an event log and a CPN. It allows to quantify to which extent the behavior seen in the trace complies with the CPN model. The metric is based on a proportion of the total number of token jumps $\texttt{j}$ and tokens transferred $\texttt{k}$.

**Definition 4 (Fitness).** *Let $\sigma$ be a trace, and let $CP$ be a colored Petri net. Let $\texttt{j}$ be the total number of token jumps and, let $\texttt{k}$ be the total number of tokens transferred, computed in Algorithm 1 with $\sigma$ and $CP$ as input. Then, the (global) fitness metric $\texttt{fit}(\sigma, CP)$ is defined as:*

$$\texttt{fit}(\sigma, CP) = 1 - \frac{\texttt{j}}{\texttt{k}}$$

We shall demonstrate that $0 \leq \texttt{fit}(\sigma, CP) \leq 1$. Let us focus on counter $\texttt{k}$. In each event $e$, we transfer $|R(e)|$ tokens, as we force to replay all event's objects. Also, all distinct objects are consumed at the end of the method. Thus, we have that $k = (\sum_{\forall e \in \sigma} |R(e)|) + |R(\sigma)|$ with $|R(\sigma)| > 0$. Regarding the counter $\texttt{j}$. Let $\texttt{j}_e$ be the number of token jumps made in an event $e$ of a trace $\sigma$. In every event $e$, we know that at most $|R(e)|$ jumps can be made, so $0 \leq \texttt{j}_e \leq |R(e)|$. Let $\texttt{j}_F$ be the number of token jumps of the distinct objects to sink places as they remained in intermediate places after the replay (e.g., see Lines 14-21 in Algorithm 1). We know that $0 \leq \texttt{j}_F \leq |R(\sigma)|$. Now, let us formulate the total number of jumps as $\texttt{j} = (\sum_{\forall e \in \sigma} \texttt{j}_e) + \texttt{j}_F$. Then, it follows that $\texttt{j} \leq \texttt{k}$. Therefore, $0 \leq \texttt{fit}(\sigma, CP) \leq 1$.

We now extend the definition of fitness for an event log as the average of the fitness values, which are computed upon the replay of each trace in the event log on top of a colored Petri net.

**Definition 5 (Fitness of an event log).** *Let $L$ be an event log, and let $CP$ be a colored Petri net. We denote by $\texttt{fit}(L, CP)$ the average fitness value obtained upon replaying individually every trace $\sigma$ on top of $CP$ using Algorithm 1, where:*

$$\texttt{fit}(L, CP) = \frac{1}{|L|} \sum_{\forall \sigma \in L} \texttt{fit}(\sigma, CP)$$

For example, let us consider the replay of traces $\sigma_1$ and $\sigma_2$ of Table 1 on top of the CPN of Fig. 1. After the execution of Algorithm 1 with $\sigma_1$, the obtained fitness value is $\texttt{fit}(\sigma_1, CP) = 1 - \frac{0}{9} = 1$, whereas with $\sigma_2$, we have that $\texttt{fit}(\sigma_2, CP) = 1 - \frac{4}{10} = 0.6$. These global measures may be interpreted as follows: the overall system's behavior observed in $\sigma_1$ complies completely with the specification model. Conversely, $\texttt{fit}(\sigma_2, CP) = 0.6$ indicates that only 60% of the overall behavior observed in $\sigma_2$ complied with specification model. Then, by considering both traces of the log using Definition 5, $\texttt{fit}(L, CP) = 0.8$ gives an estimation on how in average the system, as observed in the logs, complies the specification model.

### 5.2. Local Conformance Diagnostics

In the previous part of this section, we introduced a global conformance measure, namely fitness (Definitions 4 and 5). This measure allows to quantify to which extent the real system, as observed in logs, comply with the specification. Whilst such a metric provides an overall compliance estimation for the whole system, in many applications is required to provide *local diagnostics*, i.e., in which precise system components deviations are occurring, and in which magnitude.

In this part, we present *local conformance metrics* that are related to precise components of a system, and which can be computed upon the execution of our conformance checking method. Our approach is based on the direct association between real components of a system and components of a CPN input model. Recall that activities correspond to transitions, real locations to places, and the relation between a location and an activity is represented by an arc. Thus, by keeping track of the proportion of token transfers and jumps flowing through a model component, we can precisely indicate the number and magnitude of deviations occurred in the part of the real system that such model component represents. In what follows, we introduce these metrics, and we illustrate their usage in an example.

**Definition 6** (**Place-conformance**). *Let $\sigma$ be a trace, and let $p \in P$ be a place in a CPN. Let $\mathtt{k}_\sigma(p)$ be the number of tokens consumed from $p$ upon the replay of $\sigma$, and let $\mathtt{j}_\sigma(p)$ denote the number of token jumps to place $p$ upon the replay of $\sigma$. The place-conformance $\mathtt{fit}_\sigma(p)$ is defined as:*

$$\mathtt{fit}_\sigma(p) = \begin{cases} 1 - \dfrac{\mathtt{j}_\sigma(p)}{\mathtt{k}_\sigma(p)} & : \quad \mathtt{k}_\sigma(p) > 0 \\ \bot & : \quad \textit{otherwise} \end{cases}$$

The *place-conformance* $\mathtt{fit}_\sigma(p)$ compares the number of tokens consumed from place $p$, when replaying trace $\sigma$, with how many of them actually jumped to $p$ to force transition firings. Hence, this metric can be seen as the proportion of objects that comply with be at the location represented by place $p$ upon the execution of any activity that requires objects from such a location. If $\mathtt{fit}_\sigma(p)$ is close to 1, then almost no tokens jumped to $p$, e.g., objects are respecting the path established in the specification. Conversely, if $\mathtt{fit}_\sigma(p)$ is close to 0, then most of the tokens are jumping to place $p$, e.g., the majority of the objects skip previous activities that precede the location represented by $p$. Note that if $\mathtt{k}_\sigma(p) = 0$, then $\mathtt{fit}_\sigma(p)$ is not defined, i.e., assessments cannot be computed since no traffic flowed through place $p$ during the replay of $\sigma$.

**Definition 7** (**Flow-conformance**). *Let $\sigma$ be a trace, and let $(p, t) \in F$ be an input arc in a CPN, s.t. $p \in P \wedge t \in T$. Let $\mathtt{k}_\sigma(p, t)$ be the number of tokens consumed from $p$ to fire $t$ when replaying trace $\sigma$, and let $\mathtt{j}_\sigma(p, t)$ denote the number of tokens that jumped to place $p$ to force the firing of $t$. The flow-conformance $\mathtt{fit}_\sigma(p, t)$ is defined as:*

$$\mathtt{fit}_\sigma(p, t) = \begin{cases} 1 - \dfrac{\mathtt{j}_\sigma(p, t)}{\mathtt{k}_\sigma(p, t)} & : \quad \mathtt{k}_\sigma(p, t) > 0 \\ \bot & : \quad \textit{otherwise} \end{cases}$$

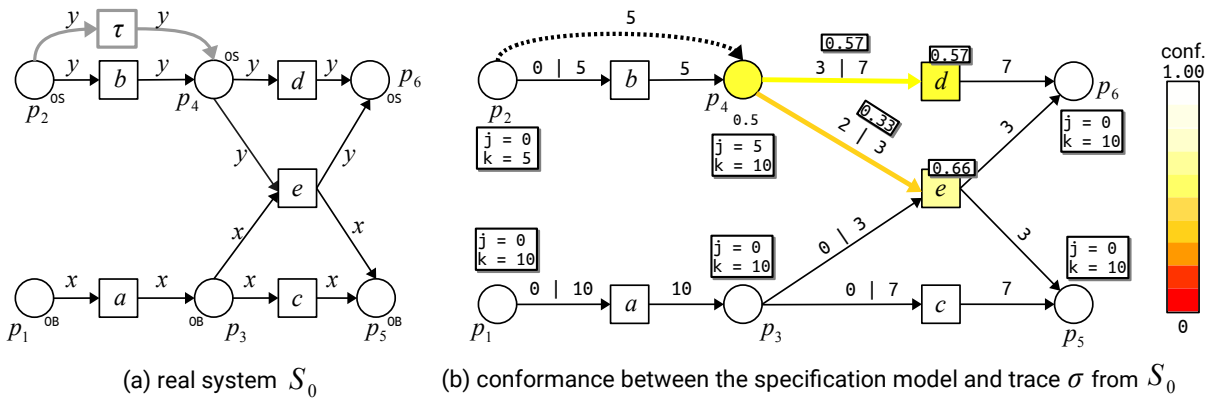
**Definition 8** (**Transition-conformance**). *Let $\sigma$ be a trace, and let $t \in T$ be a transition in a CPN. Let us define the active pre-set of transition $t$ in trace $\sigma$ as ${}^\bullet t_\sigma = \{p \mid p \in {}^\bullet t \wedge \mathtt{k}_\sigma(p, t) > 0\}$. The transition-conformance $\mathtt{fit}_\sigma(t)$ is defined as:*

$$\mathtt{fit}_\sigma(t) = \begin{cases} \dfrac{1}{|{}^\bullet t_\sigma|} \displaystyle\sum_{\forall p \in {}^\bullet t_\sigma} \mathtt{fit}_\sigma(p, t) & : \quad |{}^\bullet t_\sigma| > 0 \\ \bot & : \quad \textit{otherwise} \end{cases}$$

Definitions 7 and 8 follow the same principle of place-conformance. Given the replay of a trace $\sigma$, the *flow-conformance* $\texttt{fit}_\sigma(p, t)$ compares the number of tokens transferred, through the arc from place $p$ to transition $t$, with how of many them jumped to $p$ to force specifically the firing of $t$. This can be interpreted as the proportion of objects that comply to be at the location related to place $p$ when executing activity $\Lambda(t)$.

The *transition-conformance* $\texttt{fit}_\sigma(t)$ is the mean value of the flow-conformance between $t$ and all the input places from which $t$ consumes tokens. Thus, $\texttt{fit}_\sigma(t)$ diagnoses how many of the objects consumed by activity $\Lambda(t)$, from all its required locations, correspond to outliers.

We now exemplify the usage of the local conformance metrics. Let us recall our motivating example of trading systems shown in Section 2. Fig. 6(a) shows a model representing a real trading system $S_0$ whose behavior is slightly deviated from the specification model of Fig. 1. In particular, some sell orders in $S_0$ can be submitted to the sell side of an order book (place $p_4$) by skipping activity $b$. Now, let us consider a trace $\sigma$ from $S_0$. Let us assume that $\sigma$ corresponds to the observed interaction of 20 distinct objects (i.e., 10 buy orders and 10 sell orders). We then ran our method to check conformance between $\sigma$ and the model of Fig. 1. Let us suppose $\texttt{j} = 5$ and $\texttt{k} = 55$ as the total number of jumps and token transfers computed by Algorithm 1, thereby obtaining a fitness value $\texttt{fit}(\sigma, CP) = 1 - \frac{5}{55} = 0.909$ (i.e., Definition 4). Nevertheless, it becomes more insightful to diagnose conformance in precise system components. To this aim, we calculate the local conformance metrics previously introduced and we extend the specification of the specification model of Fig. 1 with such metrics. The model is also extended with relevant traffic statistics of token jumps and transfers in the model components (i.e., see Fig. 6(b)).




(a) real system $S_0$ (b) conformance between the specification model and trace $\sigma$ from $S_0$

Fig. 6: Conformance results between a trace $\sigma$ of system $S_0$ (a) and the specification (b).

We briefly discuss the specification model of Fig. 6(b) which has been enriched with conformance diagnostics. First, the number of transferred tokens $\texttt{k}_\sigma(p)$ and token jumps $\texttt{j}_\sigma(p)$ are shown besides each place $p$, e.g., $\texttt{k}_\sigma(p_2) = 10$ and $\texttt{j}_\sigma(p_2) = 5$. Jumps between places are displayed as dotted lines labeled with their frequency. In this example, jumps corresponded to 5 sell orders that in the system $S_0$ moved to the sell side of the order book (place $p_4$) using the silent action $\tau$. This action is not recorded in trace $\sigma$ nor allowed by the specification model shown in Fig. 1. Then, upon the replay of activities $d$ and $e$, these sell orders were not in place $p_4$, but in place $p_2$. Hence, to force replay, these 5 tokens jumped from place $p_2$ to place $p_4$.

Input arcs are labeled using notation $\texttt{j}_\sigma(p, t) \mid \texttt{k}_\sigma(p, t)$ where $\texttt{k}_\sigma(p, t)$ is the number of tokens consumed by transition $t$ from place $t$, and $\texttt{j}_\sigma(p, t)$ indicates the number of jumps to place $p$ to force the firing of $t$. As an example, the label of input arc $(p_4, d)$ indicates that 7 sell orders were consumed by activity $d$, but 3 of them were not ready in place $p_4$ before the firing. Output arcs are simply labeled with the number of resources transferred from a transition to a place.

Local conformance diagnostics are displayed on components of the specification model. For example, for place $p_4$, the place-conformance $\texttt{fit}_\sigma(p_4) = 0.5$. This can be interpreted that only half of the time a sell order was ready in the sell side of the order book upon the execution of activity $d$ or $e$. The flow-conformance $\texttt{k}_\sigma(p_4, e) = 0.33$, i.e., activity $e$ consumed a sell order 3 times, but two of these orders were not ready in place $p_4$. The transition-conformance $\texttt{fit}_\sigma(e) = 0.66$ is the mean value between $\texttt{fit}_\sigma(p_3, e) = 1$ and $\texttt{fit}_\sigma(p_4, e) = 0.33$. This means that buy orders were always available in place $p_3$, whereas only a third of the time sell orders were available in place $p_4$. As a result, 66% of the time no deviations were observed upon the execution of activity $e$.

Notably, a practical benefit of the use of local conformance metrics is their combination with the notion of a *heat map*. For example, in Fig. (see Fig. 6(b)) a *heat bar* is displayed in the right side of the model denoting that the lower the conformance measure of a component, then the more red that such a component is painted. This allows us to quickly identify which components experienced more deviations. For instance, in Fig 6(b), it can be easily seen that deviations are localized in the component of the system related to sell orders.

Finally, note that the introduced local measures are computed based on the information provided by a single trace $\sigma$. We close this section by extending definitions of these measures to event logs. Considering now all traces in the log, they shall allow to diagnose the average magnitude of deviations in precise components of the system.

**Definition 9 (Place-conformance of an event log).** *Let $L$ be an event log, and let $p \in P$ be a place in a CPN. Let us define $L_p = \{\sigma \mid \sigma \in L \wedge \texttt{k}_\sigma(p) > 0\}$. The place-conformance $\texttt{fit}_L(p)$ is defined as:*

$$\texttt{fit}_L(p) = \begin{cases} \dfrac{1}{|L_p|} \displaystyle\sum_{\forall \sigma \in L_p} \texttt{fit}_\sigma(p) & : \quad |L_p| > 0 \\ \bot & : \quad \textit{otherwise} \end{cases}$$

**Definition 10 (Flow-conformance of an event log).** *Let $L$ be an event log, and let $(p, t) \in F$ be an input arc in a CPN, s.t. $p \in P \wedge t \in T$. Let us define $L_{(p,t)} = \{\sigma \mid \sigma \in L \wedge \texttt{k}_\sigma(p, t) > 0\}$. The flow-conformance $\texttt{fit}_L(p, t)$ is defined as:*

$$\texttt{fit}_L(p, t) = \begin{cases} \dfrac{1}{|L_{(p,t)}|} \displaystyle\sum_{\forall \sigma \in L_{(p,t)}} \texttt{fit}_\sigma(p, t) & : \quad |L_{(p,t)}| > 0 \\ \bot & : \quad \textit{otherwise} \end{cases}$$

**Definition 11 (Transition-conformance of an event log).** *Let $L$ be an event log, and let $t \in T$ be a transition in a CPN, s.t. $p \in P \wedge t \in T$. Let us define $L_t = \{\sigma \mid \sigma \in L \wedge \exists p : \texttt{k}_\sigma(p, t) > 0\}$. The transition-conformance $\texttt{fit}_L(t)$ is defined as:*

$$\texttt{fit}_L(t) = \begin{cases} \dfrac{1}{|L_t|} \displaystyle\sum_{\forall \sigma \in L_t} \texttt{fit}_\sigma(t) & : \quad |L_t| > 0 \\ \bot & : \quad \textit{otherwise} \end{cases}$$

## 6. Implementation and Experimental Evaluation

We have developed a prototypical implementation of our method in the Python programming language. Our solution is supported by SNAKES [19] — a library which facilitates the prototyping of high-level classes of Petri nets, including CPNs. In the following, we describe the functioning of our prototype, as well as we report an experimental evaluation of our method. The prototype and all the material of our experiment is freely available in our project repository [20].

Fig. 7 illustrates the organization of our prototypical implementation. Users of our solution simply need to invoke a program called "conformance checker". The program receives three input arguments: an option indicating the conformance method to use (e.g., replay with CPN using jumps), an event log formatted as a comma-separated value (CSV) file, and a CPN model. CPN models are built as Python scripts. This generic organization allows us to seamlessly extend our solution, incorporating other methods of our research. In addition, as depicted in Fig. 7, our prototype has an independent routine to generate artificial event logs (as defined in Definition 5) by running CPN models. When running a CPN to generate a trace, if two or more transitions are enabled in a given marking, then the routine randomly selects one of such transitions to fire.

As an example, Fig. 8 depicts the execution of our prototype in a command-line interface with the aforementioned input arguments (option `1` stands for the conformance method presented in this paper). Upon successful execution, the program generates an output folder with CSV files corresponding to: jumps detected, frequency per trace, and their average (file `jumps.csv`), traffic statistics and the local conformance metrics presented in Section 5 per each component type (files `arcs.csv`, `transitions.csv` and `places.csv`), and finally the file `traceFitness.csv` reports the total number of token jumps, token transfers and the resulting fitness per trace and their average (Definitions 4 and 5). As we motivated in Section 2, the information provided in these reports can be used to extend the specification model, so that to quickly identify observed deviations and their magnitude in concrete components of the system.

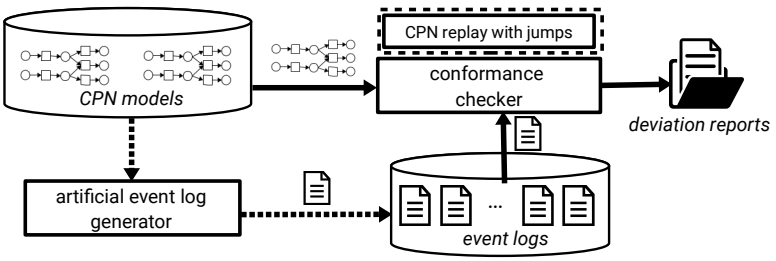



Fig. 7: Organization of the prototypical implementation of our conformance method.

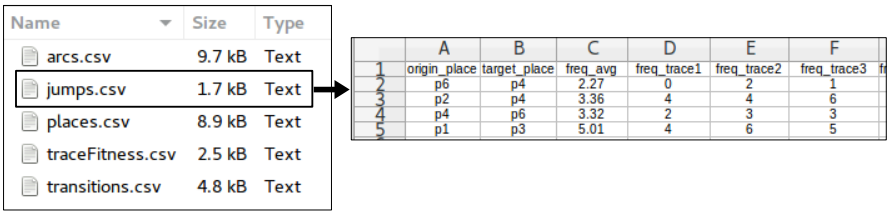
```
giulio@giulio:~/mais$ python3 conformance_checker_main.py 1 cpn_model.py event_log.csv

========== CONFORMANCE CHECKER v1.0 ==========
National Research University Higher School of Economics, Moscow, Russia.
Laboratory of Process-Aware Information Systems (PAIS Lab)
University of Constantine 2. Abdelhamid Mehri, Constantine, Algeria.
Date: May 2021
==============================================

PETRI NET MODEL: cpn_model.py
EVENT LOG: event_log.csv
CONFORMANCE METHOD: CPN Replay with Token Jumps

Replay complete! Folder with deviation reports has been generated.
```

(a) run of the prototype in the command-line interface




(b) folder with deviation reports and fragment of the file `jumps.csv`

Fig. 8: Execution of the prototype in the interface (a) and example of generated files (b).

**Experimental Evaluation**. Using our prototype, we conducted an experiment with three event logs, artificially generated from different trading system models $S_1$, $S_2$, and $S_3$. Each event log was replayed on top of the specification model of Fig. 1. These models represent replicas of a trading system, according to the specification, but each of them with undesired behavior is increasingly added. For instance, system $S_2$ is a variation of $S_1$, but with a subtle modification to slightly increase its difference with the specification model. Table 2 describes each replica and its event log generated. Each event log consists of 100 traces and 20 resources (i.e., 10 buys orders and 10 sell orders). The aim of this experiment is two-fold: to showcase the use of local conformance measures presented in Section 5, computed with all traces of an event log, and to study the stability of the proposed measures, e.g., how much the metrics are affected by subtle increases of undesired behavior.

Table 2: Experimental settings

| *event log* | *system source* | *number of events (total, avg. per trace)* | *system description (undesired behavior)* |
|---|---|---|---|
| $L_1$ | $S_1$ (Fig. 9(a)) | (2610, 26) | sell orders and buy orders skip activity $b$ and $a$. |
| $L_2$ | $S_2$ (Fig. 9(b)) | (2726, 27) | $S_1$ + activity $e$ may return sell orders to place $p_4$. |
| $L_3$ | $S_3$ (Fig. 9(c)) | (2575, 26) | $S_2$ + activity $b$ may take sell orders to a deadlock place $p_7$. |

Before discussing the conformance results, let us review the specification model in Fig. 1. According to the specification, in a system with no deviations each object must be transferred exactly 3 times: an order is submitted (activity $a$ or $b$), then it trades or is canceled (activities $c$, $d$ or $e$), and finally the order is consumed from a sink place. This implies that the replay of traces on the specification model, with 20 objects each, must count 60 token transfers (that is, 6000 transfers for an event log with 100 objects). However, as shown in Table 2, each system variant presents certain undesired behavior, so objects can move between certain locations disobeying the specification model. Hence, when replaying event logs of these system variants on top of the specification, observed deviations will incite token jumps between places, and less expected token transfers through the model structure. The latter is evidenced in columns *resources transferred* and *jumps detected* of Table 3. For ease of representation, the averages of transfers and jumps have been rounded.

Table 3 summarizes the results upon the replay of each event log on top of the specification model. For each variant, it can be observed how the introduction of a single subtle deviation induces more token jumps during replay, e.g., more objects flowing through paths unforeseen in the specification. It can be observed that the latter causes a monotonic and tenuous decrease in the average trace fitness (e.g., Definition 5). However, more intriguing becomes to identify where deviations have occurred and their magnitude. To this aim, items (d), (e), and (f) in Fig. 9 presents the specification model of Fig. 1 extended with local conformance diagnostics, computed when checking conformance in each variant, and which are associated with model components (Definitions 9—11). Input arcs and dotted lines representing jumps indicate the rounded average number of transferred/jumped tokens that flowed through them, considering all traces of a log variant. The introduction of certain undesired behavior in each variant is unveiled by our method as a jump, showing how such undesired behavior induces more deviations in a precise component. The latter impacts on the conformance-related measures, used to paint the model to clearly identify where deviations occurred more.

Table 3: Conformance results.

| *event log* | *system source* | *fitness* | *resources transferred (total, avg. per trace)* | *jumps detected (total, avg. per trace)* | *kinds of jumps (origin, target, avg. frequency per trace)* |
|---|---|---|---|---|---|
| $L_1$ | $S_1$ | 0.7974 | (4999, 50) | (1001, 10) | $(p_2, p_4, 5)$, $(p_1, p_3, 5)$ |
| $L_2$ | $S_2$ | 0.7607 | (5309, 53) | (1263, 13) | $(p_2, p_4, 5)$, $(p_1, p_3, 5)$, $(p_6, p_4, 3)$ |
| $L_3$ | $S_3$ | 0.7425 | (5058, 51) | (1306, 13) | $(p_2, p_4, 3)$, $(p_1, p_3, 5)$, $(p_6, p_4, 2)$, $(p_4, p_6, 3)$ |

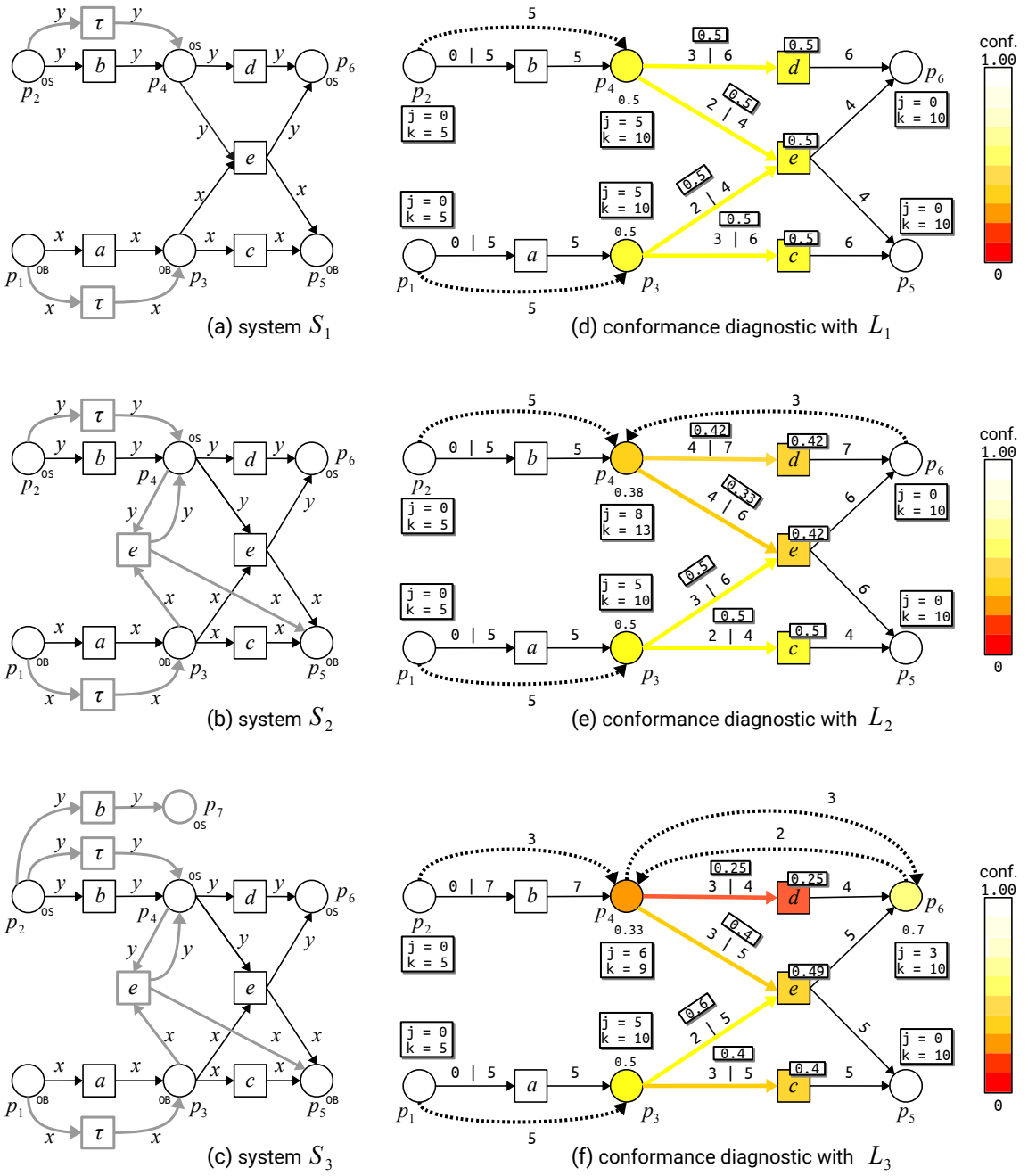



Fig. 9: Conformance checking between each event log of a system variant and the specification of Fig. 1.

.

## 7. Related Work

Conformance checking relies to a great extent on the expressive power of the models used to describe expected behavior of systems under evaluation. In this regard, state-of-the-art conformance methods use workflow nets (WF-nets) which is an ordinary Petri net employed to describe the control-flow of a process executed in isolation, that is, one single process instance after another. Thus, methods using WF-nets do not lend themselves for validating multiple instances concurrently interacting in a system. In contrast, the use of more expressive models overcoming the mentioned limitation becomes a demand in real-world applications.

For example, papers [21, 22] present case studies where multi-instance modeling notations are needed to diagnose the behavior of objects interacting within modern computational environments. In what follows, we review how different works address this challenge by using various kinds of Petri net extensions, and focusing on their application for conformance checking and similar techniques.

Proclets are one of the earliest proposals in process mining to study and validate the interaction of process instances in a system [23]. Each instance runs in an independent workflow, and semantics are provided to describe communication between workflows. Among its different applications, the concept of Proclets evolved into *artifact-centric processes* for conformance checking [24]. Also, the decomposition of the conformance checking problem has been studied for artifact-centric processes [25] in order to perform replay between each artifact and its corresponding sub-log. Another variant of this model is reported in [26], where the authors check conformance of artifacts which are modeled using UML state and activity diagrams.

Object-centric Petri nets is a notation recently proposed to describe in a single model the interaction of multiple cases (process instances) [8, 27]. Object-centric Petri nets can be seen as another subclass of CPNs with certain characteristics. For instance, arcs that transfer an arbitrary number of objects are introduced to describe one-to-many or many-to-many interactions. In [27], the authors thus present a method to discover an object-centric Petri net from event logs. In particular, these event logs are built in *extensible object-centric* (XOC) format [28]. The usage of object-centric Petri nets for conformance checking presents open challenges. For instance, the model does not provide a direct assignment of objects to concrete variables, so multiple bindings may be chosen. In this light, the fact that one event can be associated with multiple valid bindings implies the need for recursive strategies of non-linear time complexity, e.g., backtracking.

Another notable research direction in conformance checking is the validation of additional behavioral dimensions (perspectives) of a single process [14, 15] such as time or data constraints. In [14] the authors present an alignment-based conformance method using Petri nets with data (DPNs). On the one hand, alignments find differences between a model and a trace by computing the shortest path in the state space of a synchronous product net, i.e., a Petri net composed by the input model and the trace [29]. On the other hand, a DPN is a WF-net whose transitions are equipped with data variables which can be read/written upon transition firings. In this way, additional process perspectives to analyze are encoded within these variables. In contrast, DPNs are not appropriate for analysis of multiple process instances, e.g., tokens are black dots and data values do not flow through the model. This is why, for instance, we have opted for a model based on CPNs whose tokens carry object identifiers.

CPNs have already been considered in the process mining field. For instance, Rozinat et al. considered the discovery of CPN-based models for simulation [30, 31]. Notably, in paper [32], we proposed a conformance checking method with a class of CPNs whose tokens are tuples carrying object identifiers and attributes, thereby allowing to detect various kinds of deviations. For instance, using logs whose events record the state of object attributes after the event's activity execution, the method detects if such attributes were transformed as specified by the CPN. Also, the method was applied to check compliance of a real-world trading system w.r.t. its specification. In this regard, we refer the reader to papers [33–36], which present our studies on the extraction of event logs and Petri net-based modeling of real-world trading systems. However, in the method presented in [32], the replay of a trace is stopped upon the occurrence of the first deviating event, and also local diagnostics on system components are not provided. Thus, the approach of token jumps and local conformance diagnostics presented in this paper can be used to extend such methods.

Nested Petri nets is an extension where tokens can be Petri nets themselves, which allow to describe the inner behavior of objects [37]. The model becomes useful when it is crucial not only to analyze the flow of objects within a system, but also to validate the inner behavior of these objects. For instance, in paper [38], we studied the conformance problem between a nested Petri net and an event log of a multi-agent system. We proposed a compositional approach where the behavior of each agent can be checked separately.

Nested Petri Nets have also been used in the related fields of adaptive process modeling and verification [39, 40]. Finally, another recent research direction on modeling and validation of object-centric systems focuses on the use of models that combine Petri nets with data persistence models such as relational databases. The Information Systems Modeling Language (ISML) [41] and catalog-nets [42] are examples of this research direction. For instance, in these works, methods are proposed for verifying the integrity of objects in the Petri net and their representation in databases.

## 8. Conclusion

In this paper, we have presented a replay-based conformance checking method to validate whether a system, whose components manage interacting objects of different classes, complies with its specification. As the modeling language for the specification, we considered a subclass of colored Petri nets, whereas for describing real behavior we considered event logs where events are equipped with sets of involved objects. These objects refer to the individual resources involved in the execution of an activity. Regarding the subclass of CPNs, we particularly considered conservative workflow CPNs which faithfully characterizes systems handling the end-to-end processing of distinguishable objects. Among the syntactic constraints, we considered that transitions do not have input places of the same type. The latter assures in our setting only one binding for an event to replay. Consequently, we can provide an algorithm that, unlike alignments, has linear time complexity. Noteworthy to mention that the constraint of distinct types for input places can be dropped at the price of losing linear time complexity. For such a case, multiple bindings then can be associated to an event, and thus the algorithm shall look for the correct one using, e.g., recursive-based strategies such as backtracking.

The replay strategy of our method has been based on populating tokens in the model that correspond to distinct objects observed in the trace. For an event to replay, if objects are not located as tokens in specified input places of the transition to fire, we proposed a jump strategy to move tokens from their current location in the model to the required places. Interestingly, this approach not only allows us to force transition firings to find more deviations in a trace, but also recorded jumps between places unveil the so-called *desire lines*, i.e., paths of objects which are unforeseen in the specification [17].

Leveraging the fact that real locations and activities directly correspond to concrete components of a CPN, we proposed local conformance diagnostics that can be used to clearly identify the severity of deviations in precise parts of a system. Besides, we presented a prototypical implementation of our method, and we illustrated its usage with a case study on trading systems. The prototype is freely available for its usage and extension. In this regard, the prototype may be upgraded to provide automatic enhancement of an input CPN model with conformance diagnostics.

The work presented in this paper may give ground for different research directions. For instance, previous works on conformance checking based on Petri net models whose tokens carry object attributes or object inner behavior (e.g., [32, 38]) can be extended with the strategies presented in this paper, e.g., use of jumps and local conformance diagnostics. Also, it may be of interest to study other variants for this method, e.g., where tokens not only represent distinct objects, but also relationships between each other. This would imply, for instance, that the state of the objects do not correspond to a single location, but their state is in some sense distributed among places, similar to the approaches of ISML and catalog nets. The latter would make to the approach presented in this paper more intriguing and challenging.

## References

[1] W. van der Aalst, *Process Mining: Data Science in Action*, 2nd. Springer, 2016.

[2] T. Murata, “Petri nets: Properties, analysis and applications”, *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[3] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking: Relating Processes and Models*, 1st. Springer, 2018.

[4] L. Reinkemeyer, *Process Mining: Principles, Uses Cases, and Outlook*. Springer, 2020.

[5] V. Rubin, A. Mitsyuk, I. Lomazova, and W. van der Aalst, "Process Mining Can Be Applied to Software Too!", in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ACM, 2014.

[6] F. Leotta, M. Mecella, and J. Mendling, "Applying Process Mining to Smart Spaces: Perspectives and Research Challenges", in *Advanced Information Systems Engineering Workshops*, A. Persson and J. Stirna, Eds., ser. LNBIP, vol. 215, Springer, 2015, pp. 298–304.

[7] F. Mannhardt, P. Arnesen, and A. D. Landmark, "Estimating the Impact of Incidents on Process Delay", in *2019 International Conference on Process Mining (ICPM)*, IEEE, 2019, pp. 49–56.

[8] W. van der Aalst, "Object-Centric Process Mining: Dealing with Divergence and Convergence in Event Data", in *Software Engineering and Formal Methods*, P. C. Ölveczky and G. Salaün, Eds., ser. LNCS, vol. 11724, Springer, 2019, pp. 3–25.

[9] D. Fahland, "Artifact-Centric Process Mining", in *Encyclopedia of Big Data Technologies*, S. Sakr and A. Y. Zomaya, Eds. Springer, 2019, pp. 108–117. DOI: 10.1007/978-3-319-77525-8_93.

[10] J. M. E. M. van der Werf and A. Polyvyanyy, "The Information Systems Modeling Suite", in *Application and Theory of Petri Nets and Concurrency*, R. Janicki, N. Sidorova, and T. Chatain, Eds., ser. LNCS, vol. 12152, Springer, 2020, pp. 414–425.

[11] D. Fahland, "Describing Behavior of Processes with Many-to-Many Interactions", in *Application and Theory of Petri Nets and Concurrency*, S. Donatelli and S. Haar, Eds., ser. LNCS, vol. 11522, Springer, 2019, pp. 3–24.

[12] W. M. P. van der Aalst and A. Berti, "Discovering Object-centric Petri Nets", *Fundamenta informaticae*, vol. 175, no. 1/4, pp. 1–40, 2020.

[13] S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin, "Petri Nets with Parameterised Data", in *Business Process Management*, D. Fahland, C. Ghidini, J. Becker, and M. Dumas, Eds., ser. LNCS, vol. 12168, Springer, 2020, pp. 55–74.

[14] F. Mannhardt, M. Leoni, de, H. Reijers, and W. van der Aalst, *Balanced multi-perspective checking of process conformance*, ser. Computer. Springer, 2015, vol. 98, pp. 407–437.

[15] M. de Leoni and W. van der Aalst, "Data-Aware Process Mining: Discovering Decisions in Processes Using Alignments", ser. Symposium on Applied Computing (SAC 2013), ACM, 2013, pp. 1454–1461.

[16] K. Jensen and L. M. Kristensen, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*, 1st. Springer, 2009.

[17] W. van der Aalst, "Desire Lines in Big Data", in *Encyclopedia of Social Network Analysis and Mining*, R. Alhajj and J. Rokne, Eds. 2014, pp. 351–364. DOI: 10.1007/978-1-4614-6170-8_396.

[18] L. Harris, *Trading and Exchanges: Market Microstructure for Practitioners*. Oxford University Press, 2003.

[19] F. Pommereau, "SNAKES: A Flexible High-Level Petri Nets Library", in *Application and Theory of Petri Nets and Concurrency*, R. Devillers and A. Valmari, Eds., ser. LNCS, vol. 9115, Springer, 2015, pp. 254–265.

[20] Github, *Object-centric Replay-based Conformance Checking Project Repository*, https://github.com/jcarrasquel/hse-uamc-conformance-checking.

[21] G. Meroni, L. Baresi, M. Montali, and P. Plebani, "Multi-party business process compliance monitoring through IoT-enabled artifacts", *Information Systems*, vol. 73, pp. 61–78, 2018.

[22] R. Seiger, F. Zerbato, A. Burattin, L. García-Bañuelos, and B. Weber, "Towards IoT-driven Process Event Log Generation for Conformance Checking in Smart Factories", in *2020 IEEE 24th International Enterprise Distributed Object Computing Workshop (EDOCW)*, IEEE, 2020, pp. 20–26.

[23] W. van der Aalst, P. Barthelmess, C. Ellis, and J. Wainer, "Proclets: A Framework for Lightweight Interacting Workflow Processes", *International Journal of Cooperative Information Systems*, vol. 10, no. 04, pp. 443–481, 2001.

[24] D. Fahland, M. de Leoni, B. van Dongen, and W. van der Aalst, "Behavioral Conformance of Artifact-Centric Process Models", in *LNBIP*, W. Abramowicz, Ed., vol. 87, Berlin, Heidelberg: Springer, 2011, pp. 37–49.

[25] D. Fahland, M. de Leoni, B. van Dongen, and W. van der Aalst, "Conformance Checking of Interacting Processes with Overlapping Instances", in *Business Process Management*, S. Rinderle-Ma, F. Toumani, and K. Wolf, Eds., ser. LNCS, vol. 6896, Springer, 2011, pp. 345–361.

[26] M. Estañol, J. Munoz-Gama, J. Carmona, and E. Teniente, "Conformance Checking in UML Artifact-Centric Business Process Models", *Software and Systems Modeling*, vol. 18, no. 4, pp. 2531–2555, 2019.

[27] W. van der Aalst and A. Berti, "Discovering Object-Centric Petri Nets", *Fundamenta Informaticae*, vol. 175, 2020.

[28] G. Li, E. G. L. de Murillas, R. M. de Carvalho, and W. van der Aalst, "Extracting Object-Centric Event Logs to Support Process Mining on Databases", in *Information Systems in the Big Data Era*, J. Mendling and H. Mouratidis, Eds., Springer, 2018, pp. 182–199.

[29] A. Adriansyah, "Aligning observed and modeled behavior", PhD thesis, Eindhoven University of Technology (TU/e), 2014.

[30] A. Rozinat, R. Mans, M. Song, and W. van der Aalst, "Discovering colored Petri nets from event logs", *International Journal on Software Tools for Technology Transfer*, vol. 10, no. 1, pp. 57–74, 2008.

[31] A. Rozinat, R. Mans, M. Song, and W. van der Aalst, "Discovering simulation models", *Information Systems*, vol. 34, no. 3, pp. 305–327, 2009.

[32] J. C. Carrasquel, K. Mecheraoui, and I. A. Lomazova, "Checking Conformance Between Colored Petri Nets and Event Logs", in *Analysis of Images, Social Networks and Texts*, W. van der Aalst, V. Batagelj, D. I. Ignatov, M. Khachay, O. Koltsova, A. Kutuzov, S. O. Kuznetsov, I. A. Lomazova, N. Loukachevitch, A. Napoli, A. Panchenko, P. M. Pardalos, M. Pelillo, A. V. Savchenko, and E. Tutubalina, Eds., ser. LNCS, vol. 12602, Springer, 2021, pp. 435–452.

[33] J. C. Carrasquel, S. Chuburov, and I. A. Lomazova, "Pre-processing Network Messages of Trading Systems into Event Logs for Process Mining", in *Tools and Methods of Program Analysis*, ser. CCIS, vol. 1288, Springer, 2021, pp. 88–100.

[34] J. C. Carrasquel, I. A. Lomazova, and I. L. Itkin, "Towards a Formal Modelling of Order-driven Trading Systems using Petri Nets: A Multi-Agent Approach", in *Modeling and Analysis of Complex Systems and Processes (MACSPro)*, I. A. Lomazova, A. Kalenkova, and R. Yavorsky, Eds., ser. CEUR, vol. 2478, 2019.

[35] J. C. Carrasquel and I. A. Lomazova, "Modelling and Validation of Trading and Multi-Agent Systems: An Approach Based on Process Mining and Petri Nets", in *Proc. of the ICPM Doctoral Consortium*, B. van Dongen and J. Claes, Eds., ser. CEUR, vol. 2432, 2019.

[36] J. C. Carrasquel, I. A. Lomazova, and A. Rivkin, "Modeling Trading Systems using Petri Net Extensions", in *Int. Workshop on Petri Nets and Software Engineering (PNSE)*, M. Köhler-Bussmeier, E. Kindler, and H. Rölke, Eds., ser. CEUR, vol. 2651, 2020.

[37] I. A. Lomazova, "Nested Petri Nets – a Formalism for Specification and Verification of Multi-Agent Distributed Systems", *Fundamenta Informaticae*, vol. 43, pp. 195–214, 2000.

[38] K. Mecheraoui, J. C. Carrasquel, and I. A. Lomazova, "Compositional Conformance Checking of Nested Petri Nets and Event Logs of Multi-Agent Systems", in *Modeling and Analysis of Complex Systems and Processes (MACSPro)*, A. Shapoval, V. Popov, and I. Makarov, Eds., ser. CEUR, vol. 2795, 2020.

[39] I. A. Lomazova, "Nested Petri Nets for Adaptive Process Modeling", in *Pillars of Computer Science: Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, A. Avron, N. Dershowitz, and A. Rabinovich, Eds. Springer, 2008, vol. 4800, pp. 460–474.

[40] K. V. Hee, O. Oanea, A. Serebrenik, N. Sidorova, M. Voorhoeve, and I. Lomazova, "Checking Properties of Adaptive Workflow Nets", *Fundam. Informaticae*, vol. 79, pp. 347–362, 2007.

[41] J. M. E. M. van der Werf and A. Polyvyanyy, "The Information Systems Modeling Suite", in *Application and Theory of Petri Nets and Concurrency*, R. Janicki, N. Sidorova, and T. Chatain, Eds., ser. LNCS, vol. 12152, Springer, 2020, pp. 414–425.

[42] S. Ghilardi, A. Gianola, M. Montali, and A. Rivkin, "Petri Nets with Parameterised Data", in *Business Process Management*, D. Fahland, C. Ghidini, J. Becker, and M. Dumas, Eds., ser. LNCS, vol. 12168, Springer, 2020, pp. 55–74.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 2, 2021
journal homepage: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

# Severity Estimation of Defects on Interpretation of Eddy-Current Defectograms

E. V. Kuzmin[1], O. E. Gorbunov[2], P. O. Plotnikov[2], V. A. Tyukin[2], V. A. Bashkin[1]

DOI: 10.18255/1818-1015-2021-2-170-185

[1]P. G. Demidov Yaroslavl State University, 14 Sovetskaya str., Yaroslavl 150003, Russia.
[2]Center of Innovative Programming, NDDLab, 144 Soyuznaya str., Yaroslavl 150008, Russia.

MSC2020: 68T09
Research article
Full text in Russian

Received January 18, 2021
After revision March 5, 2021
Accepted March 10, 2021

To ensure traffic safety of railway transport, non-destructive tests of rails are regularly carried out by using various approaches and methods, including eddy-current flaw detection methods. An automatic analysis of large data sets (defectograms) that come from the corresponding equipment is an actual problem. The analysis means a process of determining the presence of defective sections along with identifying structural elements of railway tracks in defectograms. At the same time, severity estimation of defined defects is also of great interest. This article continues the cycle of works devoted to the problem of automatic recognition of images of defects and rail structural elements in eddy-current defectograms. In the process of forming these images, only useful signals are taken into account, the threshold levels of amplitudes of which are determined automatically from eddy-current data. The article is devoted to the issue of constructing severity estimation of found defects with various lengths. The construction of the severity estimation is based on a concept of the generalized relative amplitude of useful signals. A relative amplitude is a ratio of an actual signal amplitude to a corresponding threshold level of useful signals. The generalized relative amplitude is calculated by using the entropy of the half-normal distribution, which is assumed to be a model for a probability distribution of an appearance of certain relative amplitudes in an evaluated defect. Tuning up the formula for calculating severity estimation of a defect is carried out on the basis of eddy-current records of structural elements. As a reference of the most dangerous defect, the bolted rail joint is considered. It models a fracture of a rail. A reference weak defect is a flash butt weld, a defectogram of which contains signals with low amplitude values. The proposed approach to severity estimation of defects is shown by examples.

**Keywords:** nondestructive testing; eddy current testing; rail flaw detection; automated analysis of defectograms; severity estimation of defects

INFORMATION ABOUT THE AUTHORS

| | |
|---|---|
| Egor V. Kuzmin<br>correspondence author | orcid.org/0000-0003-0500-306X. E-mail: kuzmin@uniyar.ac.ru<br>Professor, Doctor of Science. |
| Oleg E. Gorbunov | orcid.org/0000-0001-6274-9971. E-mail: gorbunovoe@nddlab.com<br>General Director, PhD. |
| Petr O. Plotnikov | orcid.org/0000-0001-5687-7969. E-mail: plotnikovpo@nddlab.com<br>Production Engineer. |
| Vadim A. Tyukin | orcid.org/0000-0001-9149-7435. E-mail: tyukinva@nddlab.com<br>Head of Software Development Department. |
| Vladimir A. Bashkin | orcid.org/0000-0002-2534-1026. E-mail: bashkinva@nddlab.com<br>Professor, Doctor of Science. |

**Funding:** This work was supported by P. G. Demidov Yaroslavl State University Project № VIP-016.

**For citation**: E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, V. A. Tyukin, and V. A. Bashkin, “Severity Estimation of Defects on Interpretation of Eddy-Current Defectograms”, *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 170-185, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 2, 2021
сайт журнала: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

# Оценка степени опасности дефектов при расшифровке вихретоковых дефектограмм

Е. В. Кузьмин[1], О. Е. Горбунов[2], П. О. Плотников[2], В. А. Тюкин[2], В. А. Башкин[1]

DOI: 10.18255/1818-1015-2021-2-170-185

[1]Ярославский государственный университет им. П. Г. Демидова, ул. Советская, д. 14, г. Ярославль, 150003 Россия.
[2]ООО "Центр инновационного программирования", NDDLab, ул. Союзная, д. 144, г. Ярославль, 150008 Россия.

УДК 004.021
Научная статья
Полный текст на русском языке

Получена 18 января 2021 г.
После доработки 5 марта 2021 г.
Принята к публикации 10 марта 2021 г.

Для обеспечения безопасности движения на железнодорожном транспорте регулярно проводится неразрушающий контроль рельсов с применением различных подходов и методов, включая методы вихретоковой дефектоскопии. Актуальной задачей является автоматический анализ больших массивов данных (дефектограмм), которые поступают от соответствующего оборудования. Под анализом понимается процесс определения по дефектограммам наличия дефектных участков наряду с выявлением конструктивных элементов рельсового пути. При этом также большой интерес представляет и оценка степени опасности выявленных дефектов. Данная статья продолжает цикл работ, посвященных задаче автоматического распознавания образов дефектов и конструктивных элементов железнодорожных рельсов по вихретоковым дефектограммам. При формировании этих образов принимаются в расчет только полезные сигналы, пороговые уровни амплитуд которых определяются автоматически по вихретоковым данным. Статья посвящена задаче построения оценки степени опасности для выявленных поверхностных дефектов различной протяжённости. Построение оценки опирается на понятие обобщённой относительной амплитуды полезных сигналов. Относительная амплитуда представляет собой отношение реальной амплитуды сигнала к соответствующему пороговому уровню полезных сигналов. Обобщённая относительная амплитуда вычисляется с использованием энтропии полунормального распределения, которое предполагается модельным для распределения вероятностей появления тех или иных относительных амплитуд в оцениваемом дефекте. Настройка формулы расчёта степени опасности дефекта осуществляется на основе записей конструктивных элементов. В качестве эталонного наиболее опасного дефекта рассматривается болтовой рельсовый стык, который моделирует излом рельса. Эталонным слабым дефектом выступает электроконтактная сварка, дефектограмма которой, как правило, содержит сигналы с невысоким значением амплитуд. Предложенный подход к оценке степени опасности дефектов демонстрируется на примерах.

**Ключевые слова:** неразрушающий контроль рельсов; вихретоковая дефектоскопия; обнаружение дефектов; автоматический анализ дефектограмм; оценка опасности дефектов

## ИНФОРМАЦИЯ ОБ АВТОРАХ

| | |
|---|---|
| Егор Владимирович Кузьмин<br>автор для корреспонденции | orcid.org/0000-0003-0500-306X. E-mail: kuzmin@uniyar.ac.ru<br>профессор, доктор физ.-мат. наук. |
| Олег Евгеньевич Горбунов | orcid.org/0000-0001-6274-9971. E-mail: gorbunovoe@nddlab.com<br>генеральный директор, канд. физ.-мат. наук. |
| Петр Олегович Плотников | orcid.org/0000-0001-5687-7969. E-mail: plotnikovpo@nddlab.com<br>инженер-технолог. |
| Вадим Александрович Тюкин | orcid.org/0000-0001-9149-7435. E-mail: tyukinva@nddlab.com<br>руководитель сектора разработки. |
| Владимир Анатольевич Башкин | orcid.org/0000-0002-2534-1026. E-mail: bashkinva@nddlab.com<br>профессор, доктор физ.-мат. наук. |

**Финансирование:** Работа выполнена в рамках инициативной НИР ЯрГУ им. П. Г. Демидова № VIP-016.

**Для цитирования**: E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, V. A. Tyukin, and V. A. Bashkin, "Severity Estimation of Defects on Interpretation of Eddy-Current Defectograms", *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 170-185, 2021.

## Введение

Для обеспечения безопасности движения на железнодорожном транспорте регулярно проводится неразрушающий контроль рельсов с применением различных подходов и методов, включая методы вихретоковой дефектоскопии. Актуальной задачей является автоматический анализ [1—3] больших массивов данных (дефектограмм), которые поступают от соответствующего оборудования. Под анализом понимается процесс определения по дефектограммам наличия дефектных участков наряду с выявлением конструктивных элементов рельсового пути. При этом также большой интерес представляет оценка степени опасности выявленных дефектов.

Данная статья продолжает цикл работ [4—8], посвященных задаче автоматического распознавания образов дефектов и конструктивных элементов железнодорожных рельсов по дефектограммам многоканальных вихретоковых дефектоскопов. Дефектограммы разбиваются на фрагменты (блоки анализа), каждый из которых обрабатывается отдельно. В текущем блоке анализа с использованием алгоритма из статей [4—6] для каналов данных происходит выделение полезных сигналов, которые группируются в отметки. Найденные отметки подлежат дальнейшей классификации с применением нейронных сетей. В статье [7] решалась задача распознавания записей небольших конструктивных элементов (длиной до 157 мм) следующих трёх типов: 1) болтовой стык с прямым или скошенным соединением рельсов, 2) электроконтактная сварка рельсов и 3) алюминотермитная сварка рельсов. В статье [8] проводилось распознавание записей длинных (от 420 мм до 3220 мм) конструктивных элементов рельсового пути двух классов: 1) счётчик осей подвижного состава, 2) пересечение рельсовых путей. Отметки, которые не были отнесены к тому или иному типу конструктивных элементов, классифицируются как условные дефекты.

Эта статья посвящена задаче построения оценки степени опасности для выявленных поверхностных дефектов различной протяжённости. Построение оценки опирается на понятие обобщённой относительной амплитуды полезных сигналов. Относительная амплитуда представляет собой отношение реальной амплитуды сигнала к соответствующему пороговому уровню полезных сигналов. Обобщённая относительная амплитуда вычисляется с использованием энтропии полунормального распределения, которое предполагается модельным для распределения вероятностей появления тех или иных относительных амплитуд в оцениваемой отметке.

Настройка формулы расчёта степени опасности отметки осуществляется на основе записей конструктивных элементов. В качестве эталонного наиболее опасного дефекта рассматривается болтовой рельсовый стык, который моделирует излом рельса. Эталонным слабым дефектом выступает электроконтактная сварка, дефектограмма которой, как правило, содержит сигналы с невысоким значением амплитуд.

В статье рассматривается обобщённое устройство в виде 12-разрядного вихретокового дефектоскопа с 15 каналами данных (на один рельс). Каналы данных соответствуют физическим датчикам, которые последовательно располагаются на поверхности рельса перпендикулярно направлению движения дефектоскопа. Значения амплитуд сигналов каждого канала регистрируются дефектоскопом виде целых чисел от -2048 до 2047. Шаг сканирования дефектоскопа — 1 мм.

Интерес представляют амплитуды только полезных сигналов. Отметим, что применяемый ранее алгоритм определения порогового уровня амплитуд полезных сигналов [5, 6], рассчитанный на фрагменты вихретоковых дефектограмм, подавляющее большинство сигналов в которых составляет рельсовый шум, был скорректирован в работе [4]. Были учтены ситуации с возможным наличием на анализируемом фрагменте вихретоковой дефектограммы большого количество полезных сигналов (например, от протяженных поверхностных дефектов, длина которых может составлять несколько сотен метров).

Далее в статье предполагается, что полезные сигналы определяются именно по соответствующим скорректированным пороговым уровням.

## 1. Предварительные сведения

Поскольку далее в статье предполагается, что вероятности относительных амплитуд сигналов произвольной отметки имеют полунормальное распределение, а ключевым моментом при расчёте степени опасности отметки является применение энтропии, дадим сначала необходимые предварительные сведения, касающиеся используемых понятий.

Пусть непрерывная случайная величина $X$ имеет полунормальное распределение вероятностей (half-normal distribution). Тогда функция плотности распределения вероятностей имеет вид

$$f(x) = \frac{\sqrt{2}}{\sigma\sqrt{\pi}}\, e^{-\frac{x^2}{2\sigma^2}},\ x \geqslant 0,$$

где $\sigma$ — среднеквадратическое отклонение случайной величины относительно нуля.

Для полунормального распределения имеем

$$\int_0^{+\infty} f(x)\,dx = \int_0^{+\infty} \frac{\sqrt{2}}{\sigma\sqrt{\pi}}\, e^{-\frac{x^2}{2\sigma^2}}\,dx = 1,$$

$$\int_0^{+\infty} x^2 f(x)\,dx = \int_0^{+\infty} x^2 \frac{\sqrt{2}}{\sigma\sqrt{\pi}}\, e^{-\frac{x^2}{2\sigma^2}}\,dx = \sigma^2.$$

Рассмотрим формулу энтропии полунормального распределения вероятностей, записанную с использованием натурального логарифма:

$$H[X] = -\int_0^{+\infty} f(x)\,\ln f(x)\,dx = -\int_0^{+\infty} f(x)\left(\ln\frac{\sqrt{2}}{\sigma\sqrt{\pi}} - \frac{x^2}{2\sigma^2}\,\ln e\right)dx =$$

$$= \ln\frac{\sigma\sqrt{\pi}}{\sqrt{2}}\int_0^{+\infty} f(x)\,dx + \frac{1}{2\sigma^2}\int_0^{+\infty} x^2 f(x)\,dx = \ln\frac{\sigma\sqrt{\pi}}{\sqrt{2}} + \frac{1}{2} =$$

$$= \ln\sigma + \ln\frac{\sqrt{\pi}}{\sqrt{2}} + \frac{1}{2} \approx \ln\sigma + 0{,}726.$$

Отсюда получаем следующую формулу вычисления $\sigma$ через энтропию

$$\sigma \approx e^{H[X]-0{,}726}.$$

## 2. Расчет степени опасности дефекта

*Шаг 0. Нормирование амплитуд полезных сигналов.* С помощью следующей функции, написанной на языке Python 3, происходит подготовка исходных данных для расчёта степени опасности найденной отметки. Значение амплитуды каждого сигнала отметки делится на соответствующий пороговый уровень полезных сигналов и округляется с точностью до 1/*coef*, где *coef* = 10. Из полученного результата вычитается единица, т. е. осуществляется смещение на 1. Затем подсчитывается частота каждого различного значения таких относительных амплитуд со смещением.

В этой функции EC[0:49999,0:14] — это матрица анализируемых исходных вихретоковых данных, т. е. фрагмент дефектограммы длиной 50 метров с данными от 15 каналов, Threshold[0:14] — список положительных и отрицательных пороговых значений амплитуд полезных сигналов для 15 каналов, start и end — координаты начала и конца текущей отметки. Результатом работы функции является массив $R$ частот амплитуд полезных сигналов отметки, нормированных относительно соответствующего порогового уровня сигналов с последующим вычитанием единицы, умножением

на *coef* и округлением до ближайшего целого. Другими словами, элемент $R[i]$ представляет собой количество найденных значений $i = \text{round}(coef \cdot k)$, где $k$ — это отношение полезного сигнала отметки к соответствующему пороговому уровню сигналов без единицы, а round — функция округления до ближайшего целого числа. С учётом специфики хранения данных в массиве $R$ далее будем называть $i$ просто значением смещённой относительной амплитуды полезного сигнала, а $R[i]$ — частотой появления в отметке смещённой относительной амплитуды со значением $i$.

Число 10, выбранное в качестве значения для константы *coef*, означает, что смещённые относительные амплитуды различаются с точностью до одной десятой.

```
def Data_Preparation(Threshold, EC, start, end, coef, R):
    for j in range(0, 15):
        thplus = Threshold[j].thplus
        thmins = Threshold[j].thmins
        for i in range(start, end + 1):
            ec = EC[i, j]
            if ec > thplus:
                R[int(coef * ((ec / thplus) - 1) + 0.5)] += 1
            if ec < thmins:
                R[int(coef * ((ec / thmins) - 1) + 0.5)] += 1
```

*Шаг 1. Вычисление энтропии.* Рассмотрим массив $R[0 : n]$ частотного распределения значений смещённых относительных амплитуд $X = \{0, 1, \dots, n\}$ всех полезных сигналов текущей отметки. Число $n$ — значение максимальной смещённой относительной амплитуды этой отметки. Произведём вычисление энтропии $H$ для множества $X = \{0, 1, \dots, n\}$ по формуле

$$H = -\sum_{i=0}^{n} p_i \ln p_i, \text{ где } \sum_{i=0}^{n} p_i = 1,$$

$p_i$ — вероятность появления в отметке смещённой относительной амплитуды со значением $i \in X$.

Следующая функция на языке Python 3 реализует эту формулу. Кроме набора данных $R$ и значения $n$ на вход функции подаётся суммарное значение *ctr* всех элементов массива $R$. В качестве результата функция выдаёт значение $H$ энтропии для набора данных $X$.

```
def Get_Entropy(R, n, ctr):
    H = 0
    for i in range(0, n + 1):
        if R[i] > 0:
            H += (R[i] / ctr) * math.log(ctr / R[i])
    return H
```

*Шаг 2. Среднеквадратическое отклонение.* Допуская, что вероятность появления в отметке той или иной смещённой относительной амплитуды в приближении имеет полунормальное распределение, вычислим для значений этих относительных амплитуд их среднеквадратическое отклонение от нуля через полученную энтропию $H$ по следующей формуле:

$$\sigma = \frac{e^{H-0{,}726}}{coef}.$$

Деление на коэффициент *coef* проводится с учётом особенности формирования массива $R[0 : n]$, с помощью которого осуществляется хранение частот значений смещённых относительных амплитуд полезных сигналов.

*Шаг 3. Правило трёх сигм и децибелы.* Воспользуемся правилом трёх сигм, чтобы построить обобщённый уровень относительных амплитуд полезных сигналов и переведём его в децибелы:

$$Amplitude = 20 \lg(1 + 3\sigma),$$

где *Amplitude* — обобщённый уровень относительных амплитуд в децибелах.

*Шаг 4. Оценка степени опасности.* На основе обобщённого уровня *Amplitude* относительных амплитуд, выраженного в децибелах, производится расчёт степени опасности *Severity* для текущей отметки. При этом принимаются во внимание следующие параметры: *Scale* — градация (количество пунктов) степени опасности, *minAmpl* и *maxAmpl* — наименьшее и наибольшее пороговые значения обобщённого уровня относительных амплитуд. Параметр *minAmpl* позволяет игнорировать незначительные отметки, для которых не предусмотрена даже минимальная степень опасности. Значение *maxAmpl* устанавливает порог, при превышении которого более не имеет смысла различать максимально опасные отметки. В этом случае всем им присваивается предельный уровень опасности. Наивысший уровень опасности соответствует значению *Severity* = 1. С увеличением значения *Severity* уменьшается и степень опасности отметки.

Приведённая ниже функция на языке Python 3 реализует последний этап вычисления оценки степени опасности отметки.

```
def Get_Severity(Amplitude, Scale, minAmpl, maxAmpl):
    if Amplitude < minAmpl:
        Severity = Scale + 1
    else:
        Amplitude = min(Amplitude, maxAmpl)
        Step = (Scale - 1)/(maxAmpl - minAmpl)
        Severity = Scale - math.floor((Amplitude - minAmpl) * Step)
    return Severity
```

Здесь math.floor() — функция округления числа до ближайшего наименьшего целого. Отметим, что при дальнейшем анализе все отметки, получившие значение *Severity* равное *Scale* + 1, считаются незначительными по степени опасности и удаляются из списка дефектов.

## 3. Обоснование этапов расчёта

Одним из ключевых моментов при расчёте степени опасности отметки является введение понятия обобщённой относительной амплитуды. При некотором «усреднении» обобщённая относительная амплитуда позволяет получить наглядное представление о том, во сколько раз амплитуды полезных сигналов отметки больше соответствующих найденных пороговых уровней (полезных сигналов). На практике рост значения обобщённой амплитуды регистрируемых сигналов напрямую связан со степенью изношенности поверхности катания тестируемого фрагмента рельса.

В качестве «усредняющей процедуры» применяется правило трёх сигм при модельном допущении, что вероятности смещённых относительных амплитуд отметки соответствуют полунормальному закону распределения. По этому правилу предполагается, что для текущей отметки почти все относительные амплитуды без смещения на единицу ограничены сверху значением $1 + 3 \cdot \sigma$, где $\sigma$ — среднеквадратическое отклонение от нуля, вычисленное через энтропию, для значений смещённых относительных амплитуд полезных сигналов.

Важно отметить, что именно использование энтропии при вычислении среднеквадратического отклонения $\sigma$ позволяет получать стабильный ожидаемый результат при построении оценки степени опасности поверхностного дефекта. Вычисление среднеквадратического отклонения напрямую

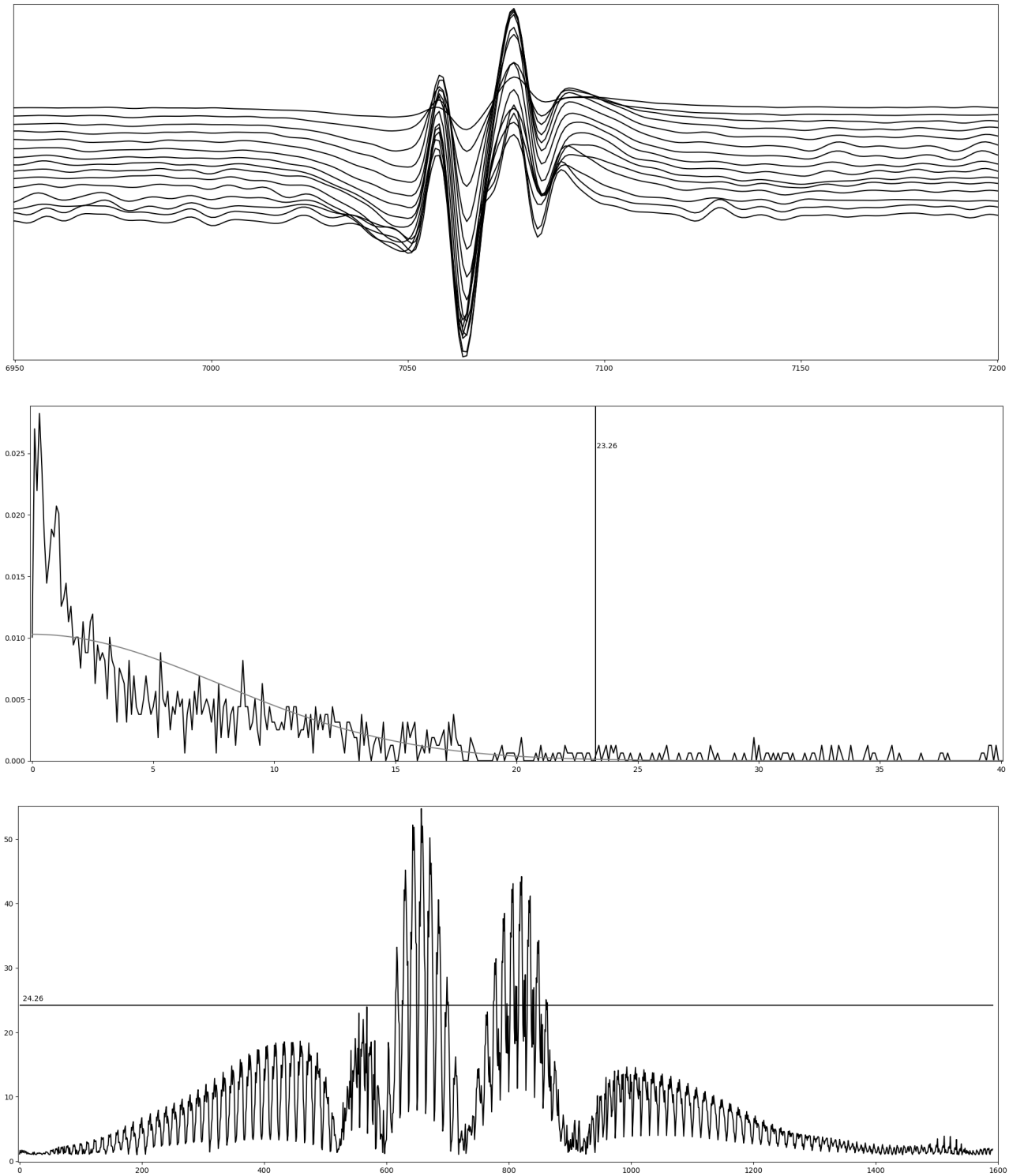



**Fig. 1.** Straight bolted joint (at the top), 250 mm, *Severity* = 1. Probability distributions of shifted relative amplitudes (at the centre). Values of relative amplitudes (at the bottom). Useful signals are 45,49%

**Рис. 1.** Прямой болтовой стык (вверху), 250 мм, *Severity* = 1. Распределения вероятностей смещённых относительных амплитуд (по центру). Значения относительных амплитуд (внизу). Полезных сигналов 45,49%

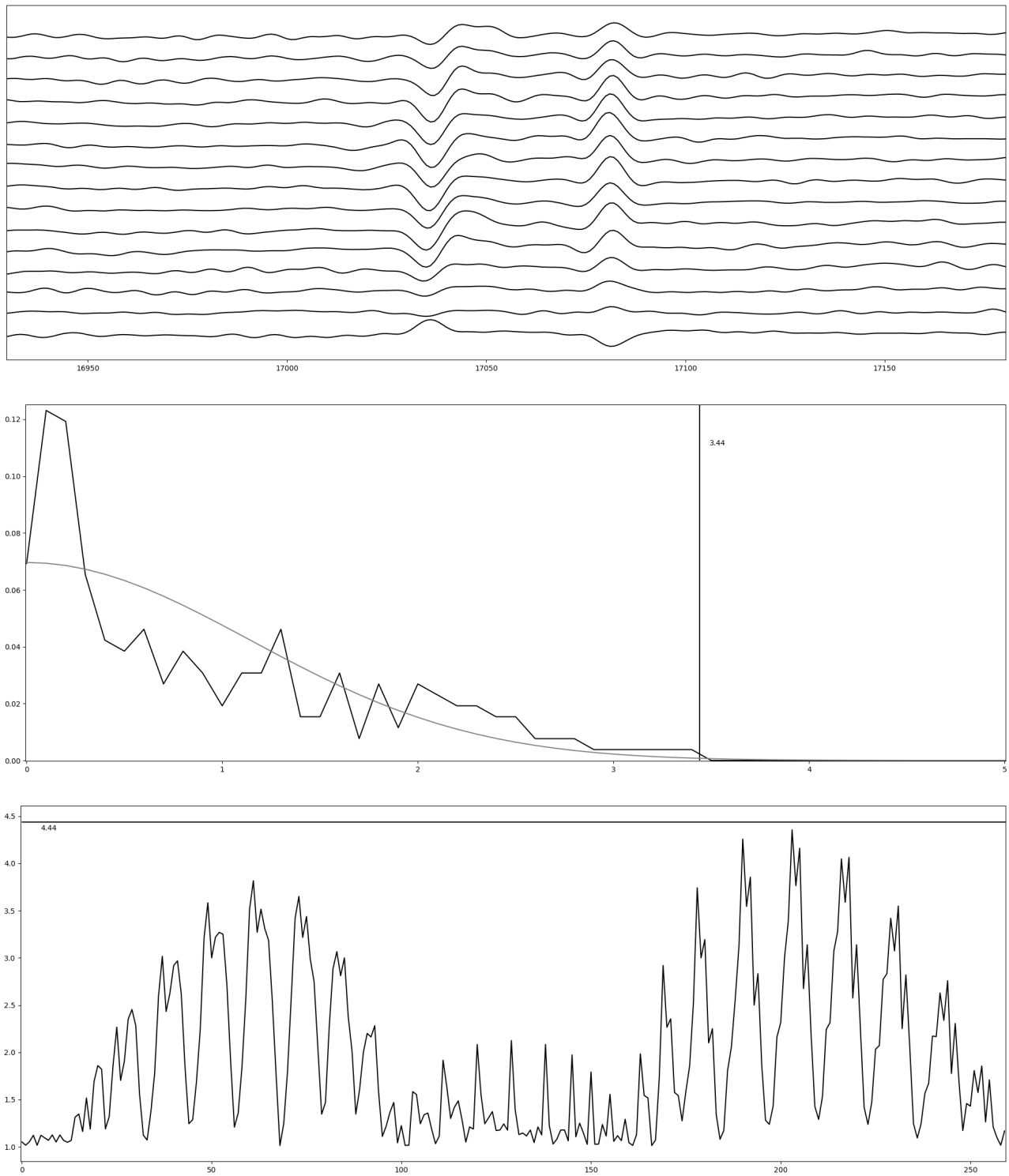



**Fig. 2.** Flash butt weld (at the top), 250 mm, *Severity* = 9. Probability distributions of shifted relative amplitudes (at the centre). Values of relative amplitudes (at the bottom). Useful signals are 7,43%

**Рис. 2.** Электроконтактная сварка (вверху), 250 мм, *Severity* = 9. Распределения вероятностей смещённых относительных амплитуд (по центру). Значения относительных амплитуд (внизу). Полезных сигналов 7,43%

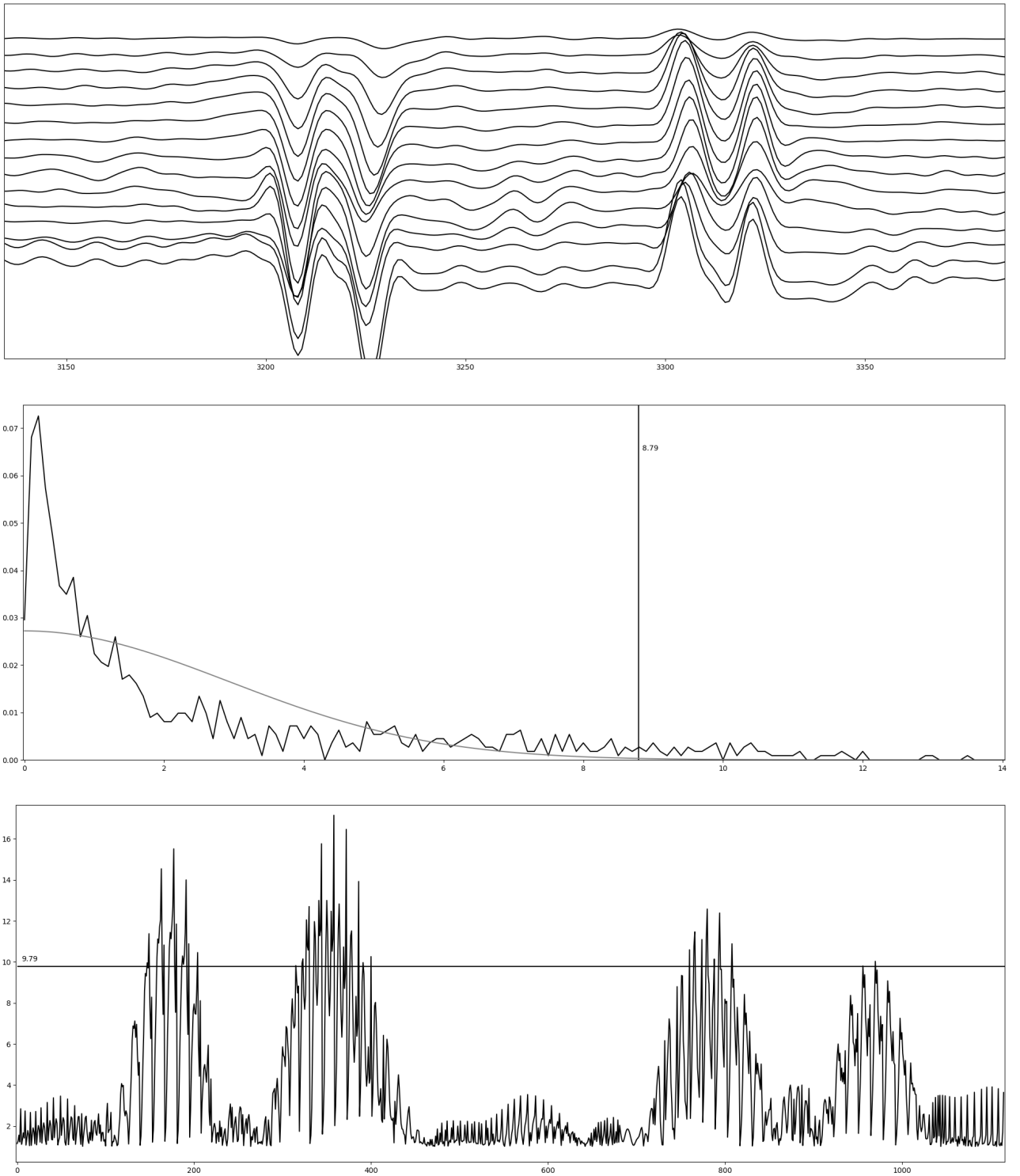



**Fig. 3.** Aluminothermic rail weld (at the top), 250 mm, *Severity* = 5. Probability distributions of shifted relative amplitudes (at the centre). Values of relative amplitudes (at the bottom). Useful signals are 31,89%

**Рис. 3.** Алюминотермитная сварка (вверху), 250 мм, *Severity* = 5. Распределения вероятностей смещённых относительных амплитуд (по центру). Значения относительных амплитуд (внизу). Полезных сигналов 31,89%

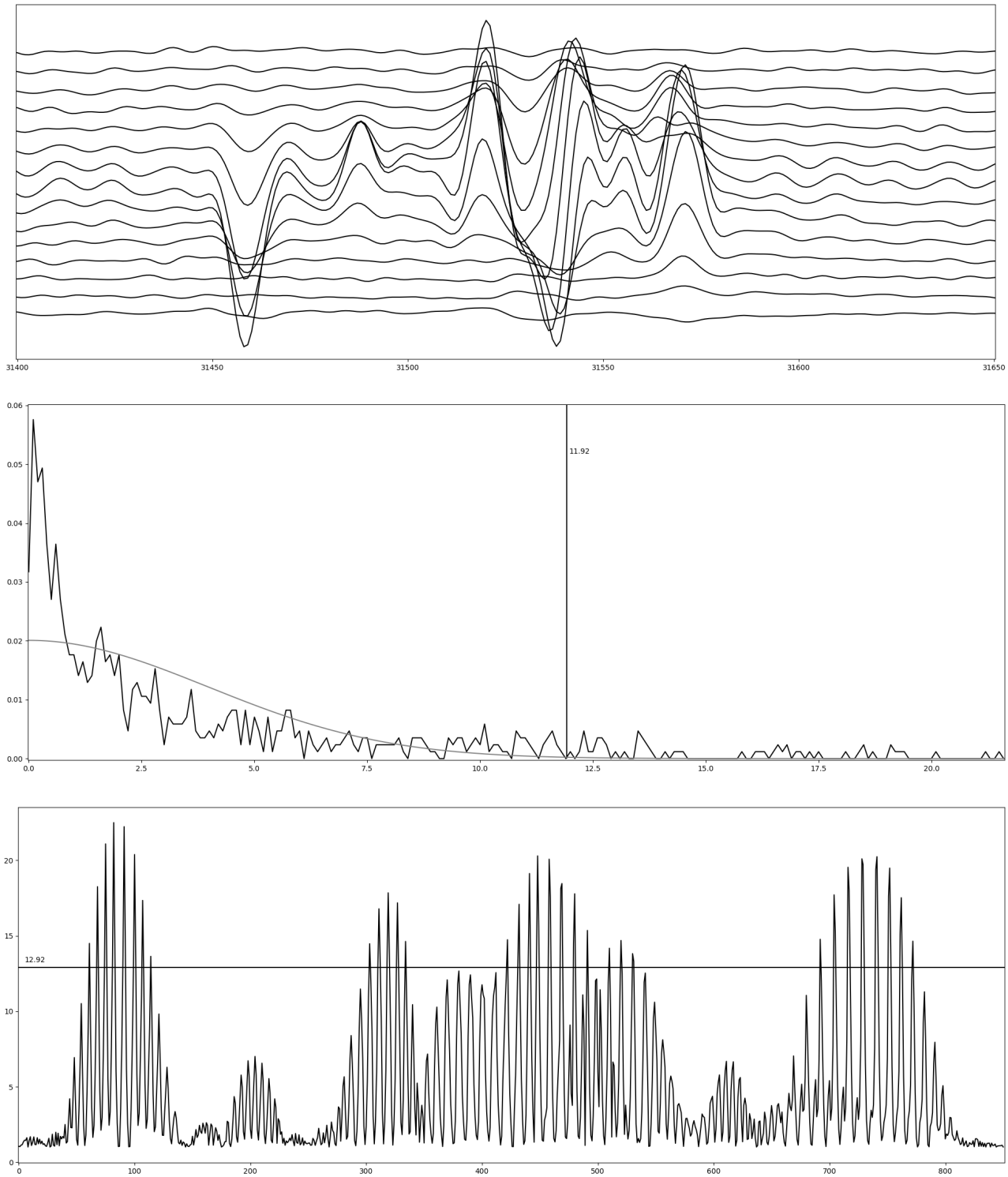



**Fig. 4.** Single surface defect (at the top), 250 mm, *Severity* = 4. Probability distributions of shifted relative amplitudes (at the centre). Values of relative amplitudes (at the bottom). Useful signals are 24,31%

**Рис. 4.** Одиночный поверхностный дефект (вверху), 250 мм, *Severity* = 4. Распределения вероятностей смещённых относительных амплитуд (по центру). Значения относительных амплитуд (внизу). Полезных сигналов 24,31%

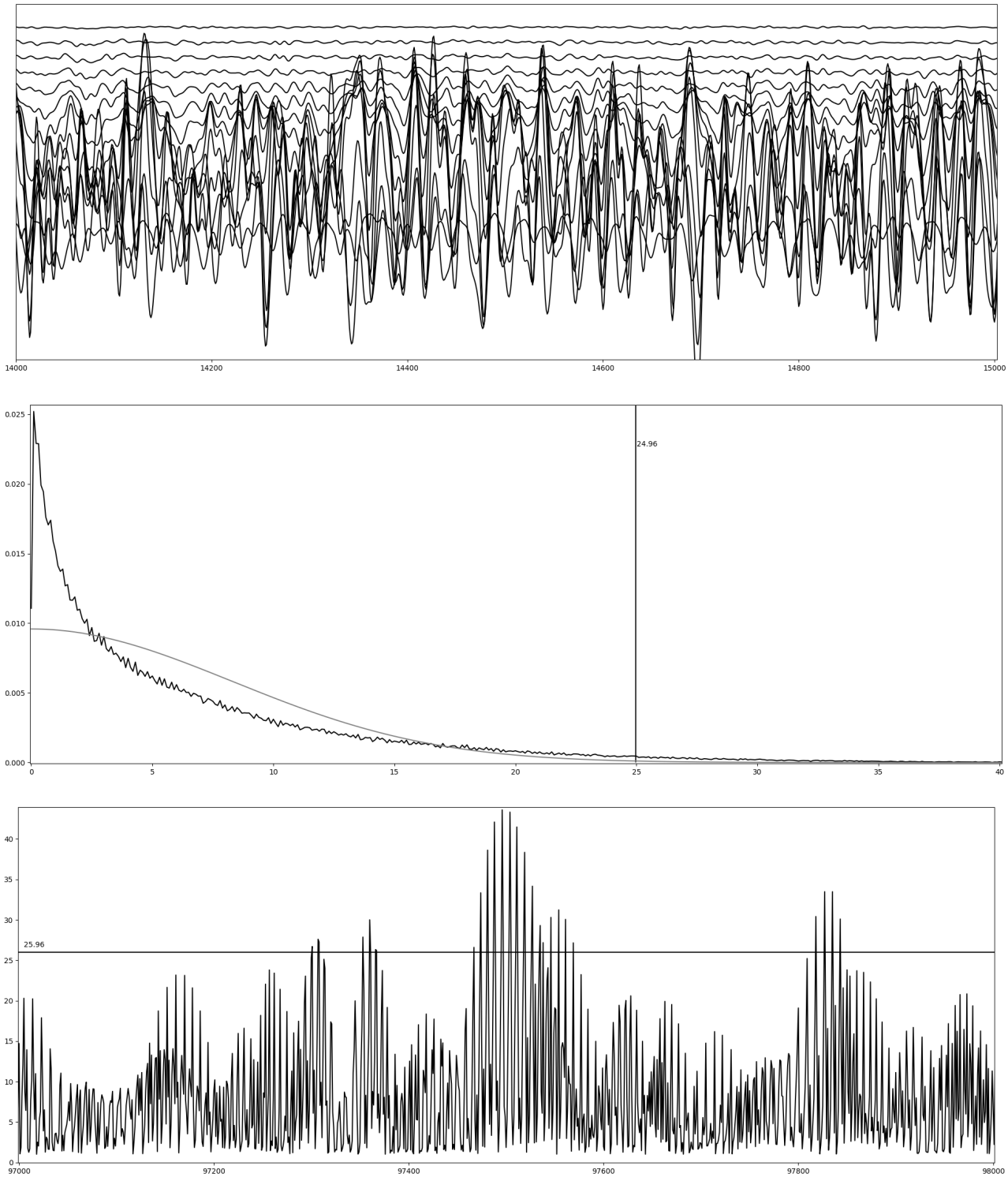



**Fig. 5.** One-meter fragment of 50-meter gauge side defect (at the top), *Severity* = 1. The first thousand values of relative amplitudes of the fragment signals (at the bottom). Useful signals are 49,74%

**Рис. 5.** Метровый фрагмент 50-метрового дефекта рабочей грани (вверху), *Severity* = 1. Первая тысяча значений относительных амплитуд сигналов фрагмента (внизу). Полезных сигналов 49,74%

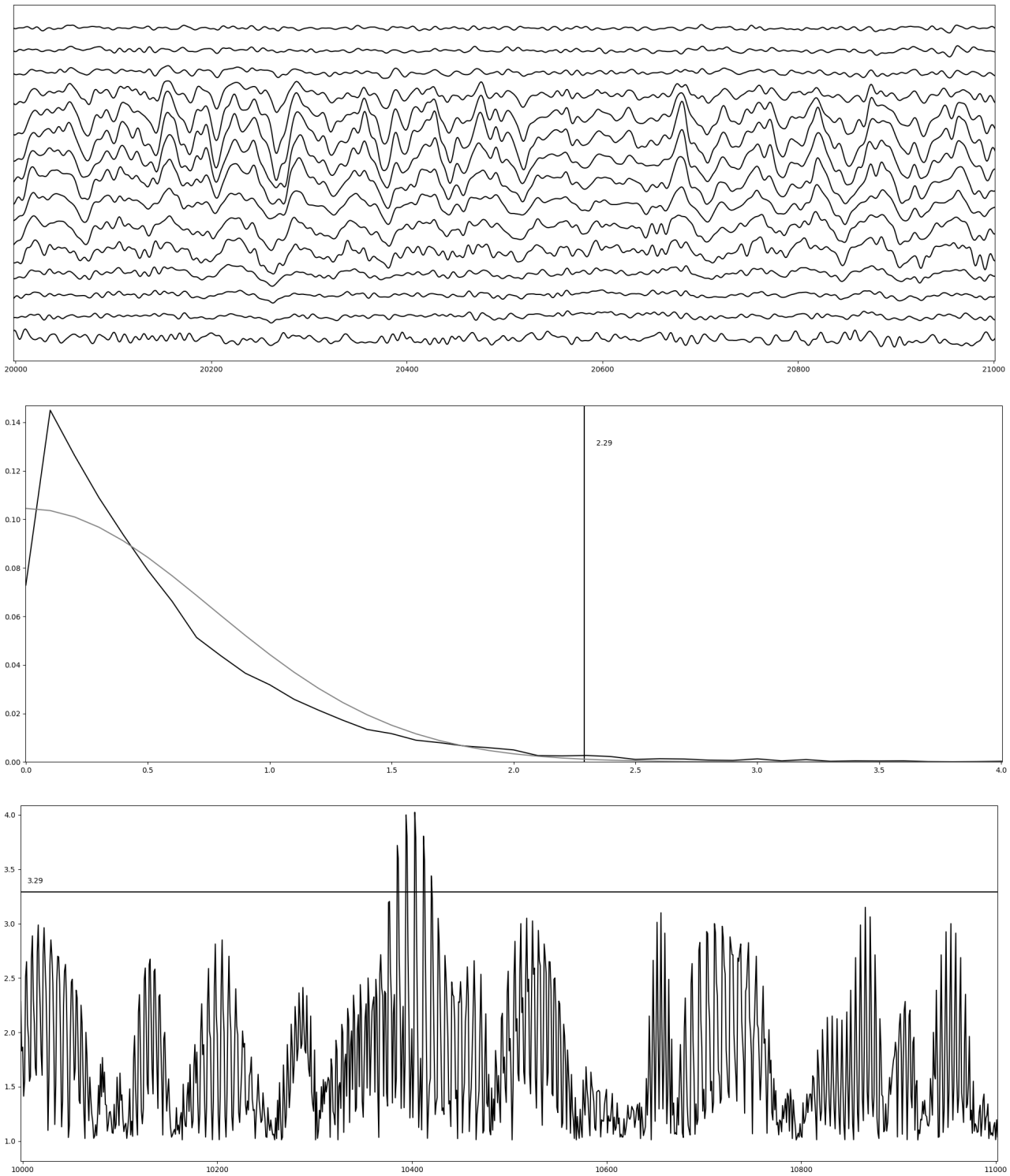



**Fig. 6.** One-meter fragment of 9-meter surface defect (at the top), *Severity* = 10. The first thousand values of relative amplitudes of the fragment signals (at the bottom). Useful signals are 13,59%

**Рис. 6.** Метровый фрагмент 9-метрового дефекта поверхности катания (вверху), *Severity* = 10. Первая тысяча значений относительных амплитуд сигналов фрагмента (внизу). Полезных сигналов 13,59%

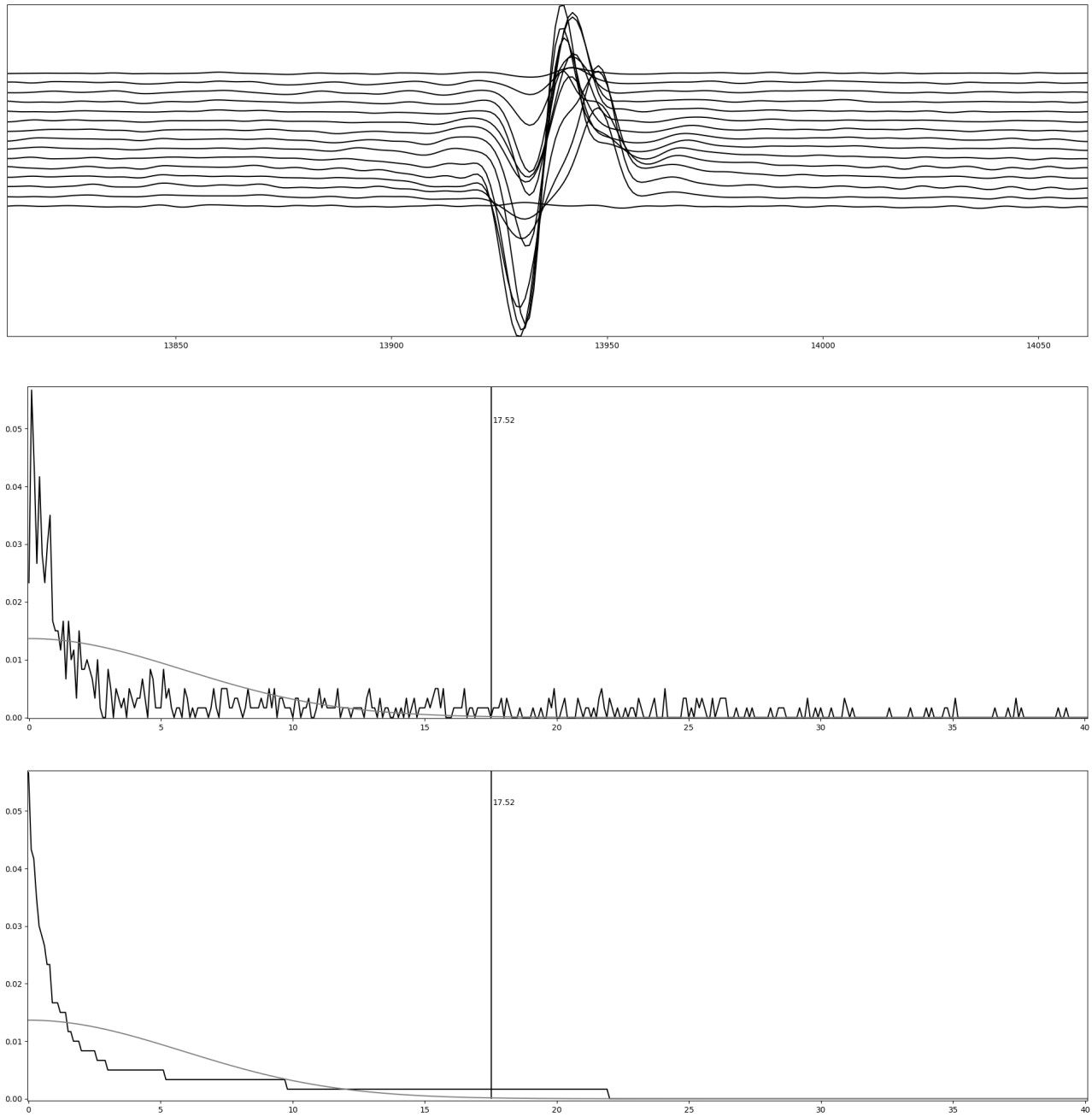



**Fig. 7.** Single short surface defect (on the top), 250 mm, *Severity* = 2. Real distribution (at the centre) and imaginary distribution (at the bottom) of probabilities of shifted relative amplitudes of useful signals. Useful signals are 17,14%

**Рис. 7.** Одиночный короткий поверхностный дефект (вверху), 250 мм, *Severity* = 2. Реальное распределение (по центру) и мнимое распределение (внизу) вероятностей смещённых относительных амплитуд полезных сигналов. Полезных сигналов 17,14%

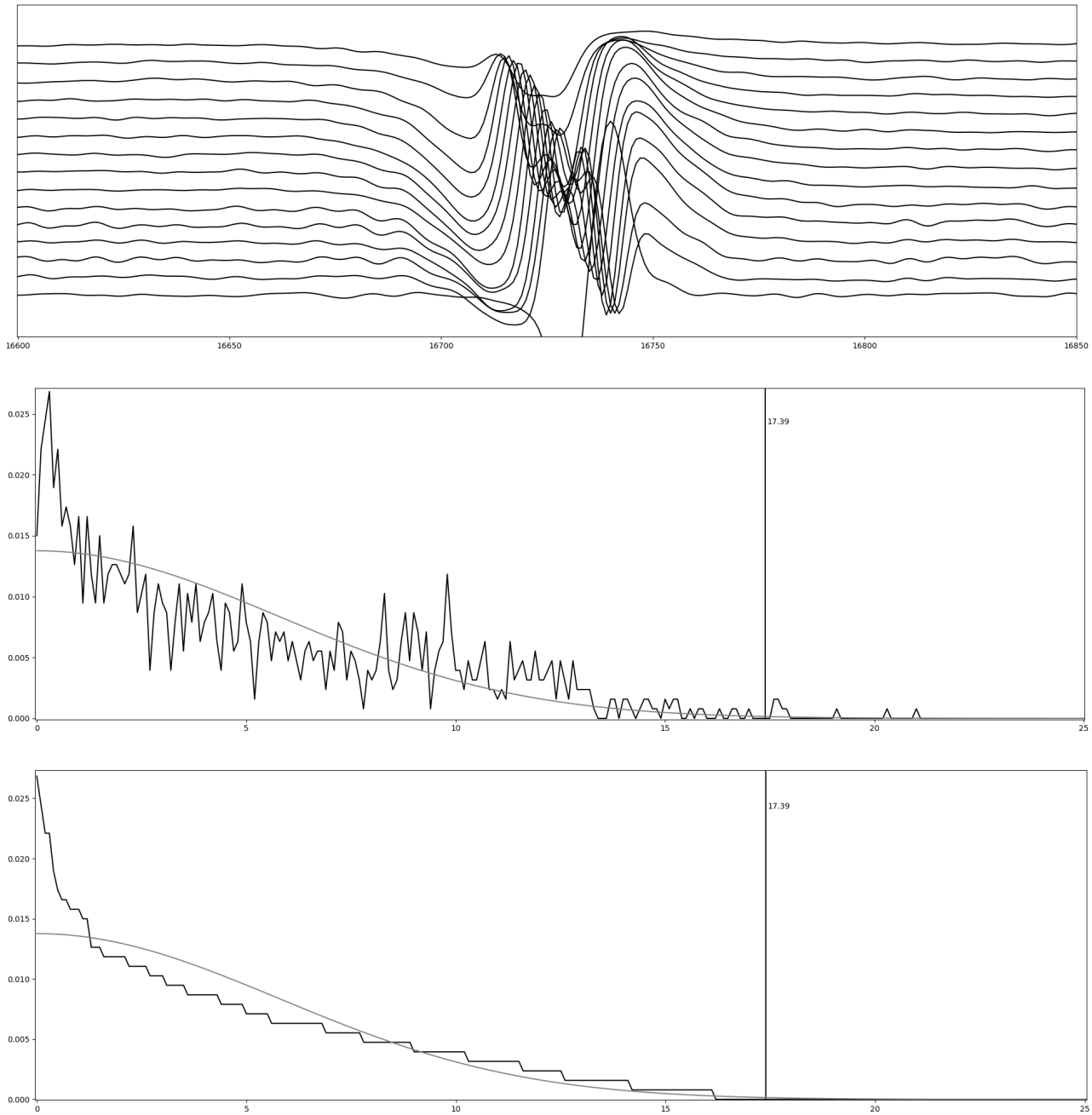



**Fig. 8.** Beveled bolted joint, 250 mm, *Severity* = 2. Real distribution (at the centre) and imaginary distribution (at the bottom) of probabilities of shifted relative amplitudes of useful signals. Useful signals are 36,20%

**Рис. 8.** Косой болтовой стык (вверху), 250 мм, *Severity* = 2. Реальное распределение (по центру) и мнимое распределение (внизу) вероятностей смещённых относительных амплитуд полезных сигналов. Полезных сигналов 36,20%

или же с применением алгоритма из статей [4—6] нецелесообразно, так как совершенно не гарантируется, что полученные значения будут согласовываться с поставленными целями по причине частого завышения или занижения (относительно ожиданий) вычисляемого значения $\sigma$.

На рис. 1–6 для различных фрагментов дефектограмм (вверху) показаны (по центру) реальное и модельное распределения вероятностей смещённых относительных амплитуд полезных сигналов с вертикальной линией отсечки для значения $3 \cdot \sigma$. Более того, в нижней части рисунков приведены выстроенные в линию значения относительных амплитуд полезных сигналов с горизонтальной линией отсечки для обобщённой относительной амплитуды, равной $1 + 3 \cdot \sigma$.

Отметим, что последующий перевод обобщённой относительной амплитуды в децибелы — это стандартное действие при работе с подобными величинами. Различное восприятие человеком близких слабых и близких сильных амплитуд приводит к необходимости промежуточного использования логарифмической шкалы для того, чтобы далее иметь возможность адекватного перехода к «линейным» уровням/степеням опасности.

Следующим ключевым моментом при построении оценки степени опасности является настройка её шкалы.

Для настройки шкалы степеней опасности предлагается использовать записи конструктивных элементов, которые регулярно встречаются на дефектограммах. В качестве эталонной отметки, имеющей максимальный уровень опасности $Severity = 1$, удобно рассматривать запись болтового рельсового стыка (см. рис. 1). В данной ситуации болтовой рельсовый стык будет играть роль излома, т. е. роль самого опасного дефекта. В качестве эталонного слабого дефекта, который имеет смысл начинать отслеживать, выступает электроконтактная сварка (см. рис. 2). Как правило, образ электроконтактной сварки формируется сигналами с относительно невысоким значением амплитуды. При таком подходе алюминотермитная сварка будет давать отметку с промежуточным уровнем опасности (см. рис. 3).

На данный момент времени настройка формулы оценки степени опасности отметки осуществлена с применением следующих значений параметров: $Scale = 10$, $minAmpl = 9$ и $maxAmpl = 27$. Оценка *Severity* для отметок, представленных на всех рисунках статьи, произведена с использованием именно этих значений параметров *minAmpl* и *maxAmpl* по десятибалльной шкале.

Наконец, обратим внимание на ещё один параметр $coef = 10$, использующийся при построении массива частот смещённых относительных амплитуд $R[0 : n]$. Этот параметр позволяет автоматически занижать степень опасности для тех дефектов, которые на записи имеют совсем не большое количество полезных сигналов. Несмотря на то что амплитуды сигналов этих дефектов могут принимать «зашкаливающие» значения, факт их небольшого количества означает, что соответствующие реальные поверхностные дефекты обладают скромными размерами и не являются настолько опасными, как может показать обобщённая относительная амплитуда.

Однако для таких отметок из-за небольшого количества полезных сигналов массив $R[0 : n]$ получается разреженным (содержит нулевые внутренние элементы). При вычислении обобщённой относительной амплитуды через энтропию наблюдается эффект сжатия массива $R[0 : n]$, поскольку он воспринимается плотным (без нулевых элементов в середине массива) и упорядоченным по убыванию. Это приводит к необходимому занижению обобщённой относительной амплитуды, по которой далее строится оценка степени опасности отметки.

Описанный эффект сжатия и сортировки набора данных $R[0 : n]$ можно наблюдать на рис. 7. Для сравнения на рис. 8 приводится похожий случай, но с достаточно плотным распределением частот амплитуд сигналов, для которого эффект сжатия почти не заметен. На этих рисунках по центру представлен реальный массив распределения частот смещённых относительных амплитуд, а в нижней части рисунков показано то, как этот массив воспринимается при вычислении энтропии (для полунормального распределения). Здесь значения элементов массива $R$ отображаются

в нормированном виде относительно общей суммы значений всех элементов массива $R$, т. е. по сути приведены графики плотности распределения вероятностей значений смещённых относительных амплитуд полезных сигналов.

## Заключение

Предложенный в статье подход к оценке степени опасности поверхностных дефектов различной протяжённости хорошо показал себя на практике при неразрушающем контроле рельсов. Алгоритм оценки успешно применяется в рамках аппаратно-программного комплекса вихретоковой дефектоскопии при оценке степени опасности дефектов поверхности катания рельсов.

## References

[1] A. A. Markov and E. A. Kuznetsova, *Rails flaw detection. Formation and analysis of signals. Book 1. Principles*. St. Petersburg: KultInformPress, 2010.

[2] A. A. Markov and E. A. Kuznetsova, *Rails flaw detection. Formation and analysis of signals. Book 2. Data interpretation*. St. Petersburg: Ultra Print, 2014.

[3] V. F. Tarabrin, A. V. Zverev, O. E. Gorbunov, and E. V. Kuzmin, "About Data Filtration of the Defectogram Automatic Interpretation by Hardware and Software Complex ASTRA", *NDT World*, vol. 64, no. 2, pp. 5–9, 2014.

[4] E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, V. A. Tyukin, and V. A. Bashkin, "An Algorithm for Correcting Levels of Useful Signals on Interpretation of Eddy-Current Defectograms", *Modeling and Analysis of Information Systems*, vol. 28, no. 1, pp. 74–88, 2021.

[5] E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, and V. A. Tyukin, "Finding the Level of Useful Signals on Interpretation of Magnetic and Eddy-Current Defectograms", *Automatic Control and Computer Sciences*, vol. 52, no. 7, pp. 658–666, 2018.

[6] E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, and V. A. Tyukin, "An Efficient Algorithm for Finding the Level of Useful Signals on Interpretation of Magnetic and Eddy Current Defectograms", *Automatic Control and Computer Sciences*, vol. 52, no. 7, pp. 867–870, 2018.

[7] E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, V. A. Tyukin, and V. A. Bashkin, "Application of Neural Networks for Recognizing Rail Structural Elements in Magnetic and Eddy Current Defectograms", *Automatic Control and Computer Sciences*, vol. 53, no. 7, pp. 628–637, 2019.

[8] E. V. Kuzmin, O. E. Gorbunov, P. O. Plotnikov, V. A. Tyukin, and V. A. Bashkin, "Application of Convolutional Neural Networks for Recognizing Long Structural Elements of Rails in Eddy-Current Defectograms", *Modeling and Analysis of Information Systems*, vol. 27, no. 3, pp. 316–329, 2020.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 2, 2021
journal homepage: www.mais-journal.ru

DISCRETE MATHEMATICS IN RELATION TO COMPUTER SCIENCE

# On Properties of a Regular Simplex Inscribed into a Ball

M. V. Nevskii[1]

DOI: 10.18255/1818-1015-2021-2-186-197

[1]P. G. Demidov Yaroslavl State University, 14 Sovetskaya str., Yaroslavl 150003, Russia.

MSC2020: 41A05, 52B55, 52C07
Research article
Full text in Russian

Received April 28, 2021
After revision May 25, 2021
Accepted May 26, 2021

Let $B$ be a Euclidean ball in $\mathbb{R}^n$ and let $C(B)$ be a space of continuos functions $f : B \longrightarrow \mathbb{R}$ with the uniform norm $\|f\|_{C(B)} := \max_{x\in B} |f(x)|$. By $\Pi_1(\mathbb{R}^n)$ we mean a set of polynomials of degree $\leq 1$, i. e., a set of linear functions upon $\mathbb{R}^n$. The interpolation projector $P : C(B) \longrightarrow \Pi_1(\mathbb{R}^n)$ with the nodes $x^{(j)} \in B$ is defined by the equalities $Pf\left(x^{(j)}\right) = f\left(x^{(j)}\right)$, $j = 1, \ldots, n+1$. The norm of $P$ as an operator from $C(B)$ to $C(B)$ can be calculated by the formula $\|P\|_B = \max_{x\in B} \sum |\lambda_j(x)|$. Here $\lambda_j$ are the basic Lagrange polynomials corresponding to the $n$-dimensional nondegenerate simplex $S$ with the vertices $x^{(j)}$. Let $P'$ be a projector having the nodes in the vertices of a regular simplex inscribed into the ball. We describe the points $y \in B$ with the property $\|P'\|_B = \sum |\lambda_j(y)|$. Also we formulate some geometric conjecture which implies that $\|P'\|_B$ is equal to the minimal norm of an interpolation projector with nodes in $B$. We prove that this conjecture holds true at least for $n = 1, 2, 3, 4$.

**Keywords:** simplex; ball; linear interpolation; projector; norm

INFORMATION ABOUT THE AUTHORS

Mikhail Viktorovich Nevskii
correspondence author
orcid.org/0000-0002-6392-7618. E-mail: mnevsk55@yandex.ru
Head of Department, Doctor of Science, Docent.

**For citation**: M. V. Nevskii, "On Properties of a Regular Simplex Inscribed into a Ball", *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 186-197, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 2, 2021
сайт журнала: www.mais-journal.ru

DISCRETE MATHEMATICS IN RELATION TO COMPUTER SCIENCE

# О свойствах правильного симплекса, вписанного в шар

М. В. Невский[1]

DOI: 10.18255/1818-1015-2021-2-186-197

[1]Ярославский государственный университет им. П. Г. Демидова, ул. Советская, д. 14, г. Ярославль, 150003 Россия.

УДК 514.17, 517.51, 519.6
Научная статья
Полный текст на русском языке

Получена 28 апреля 2021 г.
После доработки 25 мая 2021 г.
Принята к публикации 26 мая 2021 г.

Пусть $B$ — евклидов шар в $\mathbb{R}^n$, $C(B)$ — пространство непрерывных функций $f : B \to \mathbb{R}$ с равномерной нормой $\|f\|_{C(B)} := \max_{x\in B} |f(x)|$. Под $\Pi_1(\mathbb{R}^n)$ понимается совокупность многочленов от $n$ переменных степени $\le 1$, то есть линейных функций на $\mathbb{R}^n$. Интерполяционный проектор $P : C(B) \to \Pi_1(\mathbb{R}^n)$ с узлами $x^{(j)} \in B$ определяется равенствами $Pf\left(x^{(j)}\right) = f\left(x^{(j)}\right)$, $j = 1, \dots, n+1$. Норма $P$ как оператора из $C(B)$ в $C(B)$ вычисляется по формуле $\|P\|_B = \max_{x\in B} \sum |\lambda_j(x)|$, где $\lambda_j$ — базисные многочлены Лагранжа невырожденного $n$-мерного симплекса с вершинами $x^{(j)}$. Пусть $P'$ — проектор, узлы которого совпадают с вершинами правильного симплекса, вписанного в шар. В статье найдены точки $y \in B$, для которых $\|P'\|_B = \sum |\lambda_j(y)|$. Формулируется геометрическая гипотеза, из справедливости которой следует, что $\|P'\|_B$ есть минимальное значение нормы интерполяционного проектора, узлы которого принадлежат $B$. Доказывается, что эта гипотеза справедлива по крайней мере для $n = 1, 2, 3, 4$.

**Ключевые слова:** симплекс; шар; линейная интерполяция; проектор; норма

## ИНФОРМАЦИЯ ОБ АВТОРАХ

Михаил Викторович Невский
автор для корреспонденции

orcid.org/0000-0002-6392-7618. E-mail: mnevsk55@yandex.ru
Заведующий кафедрой, доктор физ.-мат. наук, доцент.

**Для цитирования**: M. V. Nevskii, “On Properties of a Regular Simplex Inscribed into a Ball”, *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 186-197, 2021.

## Введение

Пусть $\Omega$ — выпуклое тело в $\mathbb{R}^n$. Обозначим через $C(\Omega)$ пространство непрерывных функций $f : \Omega \longrightarrow \mathbb{R}$ с равномерной нормой

$$\|f\|_{C(\Omega)} := \max_{x \in Q_n} |f(x)|.$$

Под $\Pi_1(\mathbb{R}^n)$ будем понимать совокупность многочленов от $n$ переменных степени не выше 1, или, иначе говоря, линейных функций на $\mathbb{R}^n$. Для $x^{(0)} \in \mathbb{R}^n, R > 0$ через $B(x^{(0)}; R)$ обозначим $n$-мерный евклидов шар, задаваемый неравенством $\|x - x^{(0)}\| \le R$. Здесь

$$\|x\| := \sqrt{(x, x)} = \left( \sum_{i=1}^{n} x_i^2 \right)^{1/2}.$$

Положим $B_n := B(0; 1)$. Ниже $e_1, \dots, e_n$ — канонический базис $\mathbb{R}^n$.

Пусть $S$ — невырожденный симплекс в $\mathbb{R}^n$. Обозначим вершины $S$ через $x^{(j)} = \left( x_1^{(j)}, \dots, x_n^{(j)} \right)$, $1 \le j \le n + 1$. Введём в рассмотрение следующую *матрицу вершин* этого симплекса:

$$\mathrm{S} := \begin{pmatrix} x_1^{(1)} & \dots & x_n^{(1)} & 1 \\ x_1^{(2)} & \dots & x_n^{(2)} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{(n+1)} & \dots & x_n^{(n+1)} & 1 \end{pmatrix}.$$

Пусть $\mathrm{S}^{-1} = (l_{ij})$. Линейные многочлены $\lambda_j(x) = l_{1j}x_1 + \dots + l_{nj}x_n + l_{n+1,j}$, коэффициенты которых составляют столбцы матрицы $\mathrm{S}^{-1}$, обладают свойством $\lambda_j\left(x^{(k)}\right) = \delta_j^k$. Мы называем $\lambda_j$ *базисными многочленами Лагранжа,* соответствующими $S$. Для $x \in \mathbb{R}^n$ выполняются равенства

$$x = \sum_{j=1}^{n+1} \lambda_j(x) x^{(j)}, \qquad \sum_{j=1}^{n+1} \lambda_j(x) = 1.$$

Эти равенства означают, что числа $\lambda_j(x)$ являются *барицентрическими координатами* точки $x$. Подробнее см., например, [1, §1.1].

Будем говорить, что интерполяционный проектор $P : C(\Omega) \longrightarrow \Pi_1(\mathbb{R}^n)$ соответствует симплексу $S \subset \Omega$, если его узлы совпадают с вершинами этого симплекса. Проектор $P$ определяется равенствами $Pf\left(x^{(j)}\right) = f\left(x^{(j)}\right)$. Справедлив следующий аналог интерполяционной формулы Лагранжа:

$$Pf(x) = \sum_{j=1}^{n+1} f\left(x^{(j)}\right) \lambda_j(x). \tag{1}$$

Обозначим через $\|P\|_\Omega$ норму $P$ как оператора из $C(\Omega)$ в $C(\Omega)$. Из (1) следует, что

$$\|P\|_\Omega = \max_{x \in \Omega} \sum_{j=1}^{n+1} |\lambda_j(x)|.$$

Поскольку $\lambda_j(x)$ суть барицентрические координаты точки $x$, имеем также

$$\|P\|_\Omega = \max \left\{ \sum_{j=1}^{n+1} |\beta_j| : \quad \sum_{j=1}^{n+1} \beta_j x^{(j)} \in \Omega, \quad \sum_{j=1}^{n+1} \beta_j = 1 \right\}. \tag{2}$$

Пусть $\Omega = B := B(x^{(0)}; R)$. Как доказано в [2] ,

$$\|P\|_B = \max_{f_j=\pm 1} \left[ R \left( \sum_{i=1}^{n} \left( \sum_{j=1}^{n+1} f_j l_{ij} \right)^2 \right)^{1/2} + \left| \sum_{j=1}^{n+1} f_j \lambda_j(x^{(0)}) \right| \right]. \tag{3}$$

Если $S$ — правильный симплекс, вписанный в шар, то $\|P\|_B$ не зависит ни от центра $x^{(0)}$, ни от радиуса $R$ шара, ни от выбора такого симплекса. В этом случае (см. [2, теорема 2])

$$\|P\|_B = \max\{\psi(a), \psi(a+1)\}. \tag{4}$$

Здесь

$$\psi(t) := \frac{2\sqrt{n}}{n+1} \Big( t(n+1-t) \Big)^{1/2} + \left| 1 - \frac{2t}{n+1} \right|, \quad 0 \le t \le n+1, \tag{5}$$

$a := \left\lfloor \frac{n+1}{2} - \frac{\sqrt{n+1}}{2} \right\rfloor$.

Геометрические оценки в полиномиальной интерполяции изложены в монографии автора [1]. В частности, там приведены результаты по линейной интерполяции на единичном кубе $Q_n := [0,1]^n$, часть из которых имеет окончательный характер. Некоторые оценки для конкретных $n$ были позднее улучшены (см., например, [3, 4]). Интерполяция линейными функциями на евклидовом шаре в $\mathbb{R}^n$ и смежные вопросы рассматривались в [5, 6], [2].

В настоящей статье мы дополним результаты, полученные в [2], для правильного симплекса, вписанного в шар. В пункте 1 указываются точки, в которых функция $\lambda(x) := \sum |\lambda_j(x)|$ достигает своего максимума на шаре. В каждой такой точке $y \in B$

$$\lambda(y) = \sum_{j=1}^{n+1} |\lambda_j(y)| = \|P\|_B = \max\{\psi(a), \psi(a+1)\}.$$

Эти точки располагаются на граничной сфере. Их количество равно $N = \binom{n+1}{k}$, где $k$ совпадает с тем из чисел $a$ и $a+1$, на котором $\psi(t)$ принимает большее значение. Очевидно, что $N$ есть число $(k-1)$-мерных граней невырожденного $n$-мерного симплекса. В пункте 2 обсуждаются вопросы, связанные с инвариантностью нормы интерполяционного проектора при аффинном преобразовании. В пункте 3 формулируется геометрическая гипотеза, из справедливости которой следует минимальность нормы проектора с узлами в вершинах правильного вписанного симплекса.

## 1. Точки максимума функции $\lambda(x)$ для правильного вписанного симплекса

Положим для натурального $n$

$$k = k(n) := \begin{cases} a+1, & \text{если } \psi(a+1) \ge \psi(a), \\ a, & \text{если } \psi(a+1) < \psi(a). \end{cases}$$

Подробный анализ и таблица, содержащая первые значения $n$ и $k$, приведены в [2]. В таблице 1 эти данные дополняются числами $N = \binom{n+1}{k}$.

При $n = 1, 2, 3$ выполняется $k = 1$. Если $n > 3$, то $\sqrt{n+1} > 2$, поэтому

$$a + 1 = \left\lfloor \frac{n+1}{2} - \frac{\sqrt{n+1}}{2} \right\rfloor + 1 \le \frac{n+1}{2} - \frac{\sqrt{n+1}}{2} + 1 < \frac{n+1}{2}.$$

Значит, при $n > 3$ верно $k < \frac{n+1}{2}$. Так как $k$ — целое, то при всех $n \ge 2$ имеем $k \le \frac{n}{2}$.

Пусть $S$ — $n$-мерный правильный симплекс, вписанный в $n$-мерный шар $B$, $\lambda_j$ — базисные многочлены Лагранжа этого симплекса, $P : C(B) \to \Pi_1(\mathbb{R}^n)$ — соответствующий интерполяционный проектор.

**Table 1.** The numbers $n$, $k$, and $N = \binom{n+1}{k}$ **Таблица 1.** Числа $n$, $k$ и $N = \binom{n+1}{k}$

| $n$ | $k$ | $N = \binom{n+1}{k}$ |
|---|---|---|
| 1 | 1 | 2 |
| 2 | 1 | 3 |
| 3 | 1 | 4 |
| 4 | 1 | 5 |
| 5 | 2 | 15 |
| 6 | 2 | 21 |
| 7 | 3 | 56 |
| 8 | 3 | 84 |
| 9 | 3 | 120 |
| 10 | 4 | 330 |
| 11 | 4 | 495 |
| 12 | 5 | 1287 |
| 13 | 5 | 2002 |
| 14 | 6 | 5005 |
| 15 | 6 | 8008 |
| 50 | 22 | 196793068630200 |
| 100 | 45 | 110826707011209895344085355160 |

**Теорема 1.** *Рассмотрим произвольную $(k-1)$-мерную грань $G$ симплекса $S$. Пусть $H$ — $(n-k)$-мерная грань $S$, которая содержит вершины, не принадлежащие $G$. Обозначим через $g$ и $h$ центры тяжести граней $G$ и $H$. Пусть $y$ есть точка пересечения прямой $(gh)$ с граничной сферой в направлении от $g$ к $h$. Тогда*

$$\lambda(y) = \sum_{j=1}^{n+1} |\lambda_j(y)| = \|P\|_B. \tag{6}$$

*Доказательство.* Достаточно рассмотреть фиксированные шар $B \subset \mathbb{R}^n$ и правильный симплекс $S$, вписанный в $B$, а также случай $G = \operatorname{conv}\left(x^{(1)}, \dots, x^{(k)}\right)$.

Если $n = 1$, то $\psi(t) = \sqrt{t(2-t)} + |1-t|$, $a = 0$, $\psi(a) = \psi(a+1) = 1$, $k = a+1 = 1$. Возьмём $x^{(1)} = 0$, $x^{(2)} = 1$, т. е. $S = B = [0,1]$. В этом случае $g = 0$, $h = 1$, $\lambda_1(x) = -x + 1$, $\lambda_2(x) = x$, $\lambda(x) \equiv 1$. Так как $\|P\|_B = 1$, то при $y = h = 1$ равенство (6) выполняется. Заметим, что в этом тривиальном случае совокупность точек максимума функции $\lambda(x)$ совпадает с шаром $B$.

Пусть $n \geq 2$. Сначала заметим, что центр тяжести $c$ симплекса $S$ принадлежит отрезку $[g, h]$. Действительно, из равенств

$$c = \frac{1}{n+1}\sum_{j=1}^{n+1} x^{(j)}, \quad g = \frac{1}{k}\sum_{j=1}^{k} x^{(j)}, \quad h = \frac{1}{n+1-k}\sum_{j=k+1}^{n+1} x^{(j)} \tag{7}$$

следует, что $(n+1)c = kg + (n+1-k)h$, т. е.

$$c = \frac{k}{n+1}g + \frac{n+1-k}{n+1}h. \tag{8}$$

Для доказательства (6) достаточно указать линейный многочлен $p$, принимающий в узлах значения $\pm 1$, такой что $p(y) = \|P\|_B$. Равенства $p\left(x^{(j)}\right) = \pm 1$ дают

$$p(y) = \sum_{j=1}^{n+1} p\left(x^{(j)}\right)\lambda_j(y) \leq \lambda(y) = \sum_{j=1}^{n+1} |\lambda_j(y)| \leq \max_{x\in B}\sum_{j=1}^{n+1} |\lambda_j(x)| = \|P\|_B.$$

Если $p(y) = \|P\|_B$, то все величины в этой цепочке совпадают, значит, $\lambda(y) = \|P\|_B$.

Покажем, что этим свойством обладает многочлен $p \in \Pi_1(\mathbb{R}^n)$ со значениями

$$p\left(x^{(1)}\right) = \ldots = p\left(x^{(k)}\right) = -1, \quad p\left(x^{(k+1)}\right) = \ldots = p\left(x^{(n+1)}\right) = 1. \tag{9}$$

Поскольку $p$ — линейная функция, из (7) и (9) следует, что $p(g) = -1$, $p(h) = 1$. Применяя теперь (8), получаем

$$p(c) = \frac{k}{n+1}p(g) + \frac{n+1-k}{n+1}p(h) = \frac{n+1-2k}{n+1}.$$

Центр тяжести правильного симплекса совпадает с центром шара. Так как приращение функции $p$ пропорционально расстоянию между точками, выполняется

$$\frac{\|g-h\|}{p(g)-p(h)} = \frac{R}{p(y)-p(c)},$$

где $R$ — радиус шара. Подставляя найденные значения, приходим к равенству

$$p(y) = \frac{n+1-2k}{n+1} + \frac{2R}{\|g-h\|}. \tag{10}$$

Величина последней дроби не зависит от выбора шара и вписанного в него правильного симплекса. Вычислим это значение для конкретных $S$ и $B$.

Именно в качестве $S$ возьмём правильный симплекс с вершинами

$$x^{(1)} = e_1, \ \ldots, \ x^{(n)} = e_n, \quad x^{(n+1)} = \left(\frac{1-\sqrt{n+1}}{n}, \ldots, \frac{1-\sqrt{n+1}}{n}\right).$$

Длина любого ребра $S$ равна $\sqrt{2}$. Симплекс вписан в шар $B = B(x^{(0)}; R)$, где

$$x^{(0)} = \left(\frac{1-\sqrt{\frac{1}{n+1}}}{n}, \ldots, \frac{1-\sqrt{\frac{1}{n+1}}}{n}\right), \quad R = \sqrt{\frac{n}{n+1}}.$$

В соответствии с (7) координаты точек $g$ и $h$ имеют вид

$$g_1 = \ldots = g_k = \frac{1}{k}, \quad g_{k+1} = \ldots = g_n = 0,$$

$$h_1 = \ldots = h_k = \frac{1}{n+1-k} \cdot \frac{1-\sqrt{n+1}}{n},$$
$$h_{k+1} = \ldots = h_{n+1} = \frac{1}{n+1-k} \cdot \left(1 + \frac{1-\sqrt{n+1}}{n}\right).$$

Отсюда

$$\|g-h\|^2 = \left(\frac{1}{k} - \frac{1-\sqrt{n+1}}{n(n+1-k)}\right)^2 \cdot k + \left(\frac{n+1-\sqrt{n+1}}{n(n+1-k)}\right)^2 \cdot (n-k).$$

После несложных преобразований имеем

$$\|g-h\|^2 = \frac{n+1}{k(n+1-k)}.$$

Итак, в рассматриваемой ситуации

$$\frac{2R}{\|g-h\|} = 2\sqrt{\frac{n}{n+1}} \cdot \frac{(k(n+1-k))^{\frac{1}{2}}}{\sqrt{n+1}} = \frac{2\sqrt{n}}{n+1} \cdot (k(n+1-k))^{\frac{1}{2}}.$$

Продолжая теперь (10), мы можем записать

$$p(y) = \frac{n+1-2k}{n+1} + \frac{2R}{\|g-h\|} = 1 - \frac{2k}{n+1} + \frac{2\sqrt{n}}{n+1} \cdot (k(n+1-k))^{\frac{1}{2}}.$$

Если $1 \le k \le \frac{n+1}{2}$, то последнее выражение совпадает с $\psi(k)$. Это неравенство для $k$ выполняется. Более того, $k$ совпадает с тем из чисел $a$ и $a+1$, на котором $\psi(t)$ принимает большее значение. Следовательно, $p(y) = \max\{\psi(a), \psi(a+1)\} = \|P\|_B$. Теорема доказана. □

**Теорема 2.** *В обозначениях предыдущей теоремы $[g,h]$ является отрезком максимальной длины, принадлежащим $S$ и параллельным вектору $gh$.*

*Доказательство.* В [7] были получены формулы для вычисления длины и концов максимального отрезка заданного направления, принадлежащего симплексу. Можно выполнить соответствующие вычисления для конкретного симплекса и учесть соображения подобия. Однако проще воспользоваться следующей характеризацией максимального отрезка, доказанной в [7] (см. там леммы 1 и 2). *Отрезок, принадлежащий симплексу и параллельный данному ненулевому вектору, имеет максимальную длину тогда и только тогда, когда каждая $(n-1)$-мерная грань симплекса содержит хотя бы один из его концов.*

Пусть запись $x = \{\beta_1, \ldots, \beta_{n+1}\}$ означает, что точка $x$ имеет барицентрические координаты $\beta_1, \ldots, \beta_{n+1}$ относительно $S$. Через $G_j$ обозначим $(n-1)$-мерную грань симплекса, не содержащую вершину $x^{(j)}$. Для точек этой грани все барицентрические координаты неотрицательны, причём $\beta_j = 0$. Имеем:

$$g = \frac{1}{k}\sum_{j=1}^{k} x^{(j)} = \left\{\frac{1}{k}, \ldots, \frac{1}{k}, 0, \ldots, 0\right\},$$

$$h = \frac{1}{n+1-k}\sum_{j=k+1}^{n+1} x^{(j)} = \left\{0, \ldots, 0, \frac{1}{n+1-k}, \ldots, \frac{1}{n+1-k}\right\}.$$

В этих равенствах число ненулевых барицентрических координат есть соответственно $k$ и $n+1-k$. Очевидно, $g \in G_{k+1}, \ldots, G_{n+1}$, $h \in G_1, \ldots, G_k$. Поэтому каждая $(n-1)$-мерная грань симплекса $S$ содержит конец отрезка $[g,h]$. Тем самым, этот отрезок имеет максимальную длину из всех отрезков в $S$ данного направления. Заметим, что это рассуждение подходит для любого симплекса и любого $k = 1, \ldots, n$. Теорема доказана. □

## 2. Инвариантность нормы проектора при афинном преобразовании

В 1948 г. Ф. Джон [8] доказал, что каждое выпуклое тело в $\mathbb{R}^n$ содержит единственный эллипсоид максимального объёма, а также дал характеризацию тех выпуклых тел, для которых таким эллипсоидом является единичный евклидов шар $B_n$ (подробнее см., например, [9, 10]). Из теоремы Джона следует аналогичное утверждение, которое характеризует единственный эллипсоид минимального объёма, содержащий данное выпуклое тело.

Мы будем рассматривать эллипсоид минимального объёма, содержащий данный невырожденный симплекс. Для краткости такой эллипсоид будем называть *минимальным*. Очевидно, что минимальный эллипсоид симплекса описан вокруг него. Центр этого эллипсоида совпадает с центром тяжести симплекса. Минимальный эллипсоид, описанный вокруг симплекса, является евклидовым шаром тогда и только тогда этот симплекс является правильным. Это эквивалентно тому, что из всех симплексов, содержащихся в шаре, максимальный объём имеет правильный симплекс, вписанный в этот шар (см., например, [11—13]).

Положим $\varkappa_n := \mathrm{vol}(B_n)$. Через $\sigma_n$ обозначим объём правильного симплекса, вписанного в единичный шар $B_n$. Пусть $S$ — произвольный $n$-мерного симплекс и $E$ — минимальный эллипсоид этого симплекса. Если невырожденное аффинное отображение переводит $S$ в правильный симплекс, вписанный в $B_n$, то образ $E$ при этом отображении совпадает с $B_n$. Следовательно,

$$\frac{\mathrm{vol}(E)}{\mathrm{vol}(S)} = \frac{\varkappa_n}{\sigma_n}.$$

Известно, что

$$\varkappa_n = \frac{\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}+1\right)}, \qquad \sigma_n = \frac{1}{n!}\sqrt{n+1}\left(\frac{n+1}{n}\right)^{\frac{n}{2}},$$

$$\varkappa_{2m} = \frac{\pi^m}{m!}, \qquad \varkappa_{2m+1} = \frac{2^{m+1}\pi^m}{(2m+1)!!} = \frac{2(m!)(4\pi)^m}{(2m+1)!}$$

(см., например, [14, 15], [1]). Таким образом,

$$\mathrm{vol}(E) = K_n\,\mathrm{vol}(S), \qquad K_n := \frac{\varkappa_n}{\sigma_n} = \frac{n!\,(\pi n)^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}+1\right)(n+1)^{\frac{n+1}{2}}}.$$

Также справедливы равенства

$$K_{2m} = \frac{(2m)!(2\pi m)^m}{m!(2m+1)^{m+\frac{1}{2}}}, \qquad K_{2m+1} = 2^{m+\frac{1}{2}}\left(2-\frac{1}{m+1}\right)^{m+\frac{1}{2}}\pi^m m!.$$

Величина $K_n$ участвует в нижней оценке для нормы проектора, узлы которого принадлежат $B_n$. Пусть $\chi_n(t)$ есть *стандартизованный многочлен Лежандра степени n*:

$$\chi_n(t) := \frac{1}{2^n n!}\left[(t^2-1)^n\right]^{(n)}.$$

Существует константа $C > 0$, не зависящая от $n$, такая что для любого интерполяционного проектора $P : C(B_n) \longrightarrow \Pi_1(\mathbb{R}^n)$

$$\|P\|_{B_n} \geq \chi_n^{-1}(K_n) > C\sqrt{n}. \tag{11}$$

Неравенства (11) установлены автором в [6]. Правая оценка верна, например, при

$$C = \frac{\sqrt[3]{\pi}}{\sqrt{12e}\cdot\sqrt[6]{3}} = 0.2135\ldots$$

Пусть $S$ и $S'$ — невырожденные симплексы в $\mathbb{R}^n$ с вершинами $x^{(j)}, \dots, x^{(n+1)}$ и $y^{(1)}, \dots, y^{(n+1)}$ соответственно. Обозначим через $\mathbf{S}$ матрицу вершин симплекса $S$, через $\mathbf{Y}$ — $n \times (n+1)$-матрицу, $j$-й столбец которой содержит координаты вершины $y^{(j)}$. Пусть $\lambda_1, \dots, \lambda_{n+1}$ — базисные многочлены Лагранжа симплекса $S$.

**Лемма 1.** *Существует единственное аффинное преобразование $F$ пространства $\mathbb{R}^n$, переводящее $S$ в $S'$, для которого $y^{(j)} = F\left(x^{(j)}\right)$. Равенство $y = F(x)$ эквивалентно каждому из соотношений*

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \mathbf{Y}\left(\mathbf{S}^{-1}\right)^T \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{pmatrix}, \tag{12}$$

$$y = \sum_{j=1}^{n+1} \lambda_j(x) y^{(j)}. \tag{13}$$

*Доказательство.* Каждое невырожденное аффинное преобразование пространства $\mathbb{R}^n$ имеет вид $F(x) = A(x) + b$, где $A : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ — невырожденный линейный оператор. Пусть $\mathbf{A} = (a_{ij})$ — матрица оператора $A$ в каноническом базисе. В координатном виде равенство $y = A(x) + b$ эквивалентно соотношению

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} a_{11} & \dots & a_{1n} & b_1 \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & \dots & a_{nn} & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \\ 1 \end{pmatrix}.$$

Пусть $\mathbf{M}$ есть $n \times (n+1)$-матрица, стоящая в правой части. Условия $y^{(j)} = F\left(x^{(j)}\right)$ равносильны матричному равенству $\mathbf{Y} = \mathbf{M}\mathbf{S}^T$. Следовательно,

$$\mathbf{M} = \mathbf{Y}\left(\mathbf{S}^T\right)^{-1} = \mathbf{Y}\left(\mathbf{S}^{-1}\right)^T.$$

Поэтому аффинное преобразование, удовлетворяющее условиям теоремы, является единственным и имеет вид (12).

Поскольку $\lambda_j \in \Pi_1(\mathbb{R}^n)$ и $\lambda_j\left(x^{(k)}\right) = \delta_j^k$, равенство (13) также задаёт аффинное преобразование $y = F(x)$, для которого $F\left(x^{(k)}\right) = y^{(k)}$. Из единственности $F$ следует, что (13) эквивалентно (12). Равносильность этих равенств можно установить и непосредственно. Перепишем (13) в координатном виде через коэффициенты многочленов $\lambda_j$:

$$y = \sum_{j=1}^{n+1} \lambda_j(x) y^{(j)} = \sum_{j=1}^{n+1} \left( \sum_{k=1}^{n} l_{kj} x_k + l_{n+1,j} \right) y^{(j)},$$

$$y_i = \sum_{j=1}^{n+1} \left( \sum_{k=1}^{n} l_{kj} x_k + l_{n+1,j} \right) y_i^{(j)} = \sum_{j=1}^{n+1} \left( \sum_{k=1}^{n} l_{kj} x_k \right) y_i^{(j)} + \sum_{j=1}^{n+1} l_{n+1,j} y_i^{(j)} =$$

$$= \sum_{k=1}^{n} \left( \sum_{j=1}^{n+1} y_i^{(j)} l_{kj} \right) x_k + \sum_{j=1}^{n+1} y_i^{(j)} l_{n+1,j}.$$

Таким образом, (13) эквивалентно соотношениям

$$y_i = \sum_{k=1}^{n} a_{ik} x_k + b_i, \quad a_{ik} = \sum_{j=1}^{n+1} y_i^{(j)} l_{kj}, \quad b_i = \sum_{j=1}^{n+1} y_i^{(j)} l_{n+1,j}.$$

Поскольку $\mathbf{S}^{-1} = (l_{ij})$, последние равенства равносильны (12). □

Отметим, что норма интерполяционного проектора инвариантна относительно невырожденного аффинного преобразования.

**Теорема 3.** *Пусть $\Omega$ — выпуклое тело в $\mathbb{R}^n$, содержащее невырожденный симплекс $S$, $\Omega'$ и $S'$ — их образы при невырожденном аффином преобразовании, $P : C(\Omega) \longrightarrow \Pi_1(\mathbb{R}^n)$, $P' : C(\Omega') \longrightarrow \Pi_1(\mathbb{R}^n)$ — интерполяционные проекторы, узлы которых совпадают с вершинами симплексов $S$ и $S'$ соответственно. Тогда $\|P\|_\Omega = \|P'\|_{\Omega'}$.*

*Доказательство.* Пусть $x_1, \dots, x_{n+1}$ — вершины симплекса $S$. Будем считать, что вершины симплекса $S'$ занумерованы таким образом, что $y^{(j)} = F\left(x^{(j)}\right)$. При таком соответствии набор барицентрических координат произвольной точки $x \in \mathbb{R}^n$ относительно симплекса $S$ совпадает с набором барицентрических координат точки $y = F(x)$ относительно симплекса $S'$. Это следует из равенств

$$x = \sum_{j=1}^{n+1} \lambda_j(x) x^{(j)}, \quad y = \sum_{j=1}^{n+1} \lambda_j(x) y^{(j)},$$

второе из которых совпадает с соотношением (13) леммы 1. Поэтому в соответствии с формулой (2), выражающей норму интерполяционного проектора через барицентрические координаты, имеем

$$\|P\|_\Omega = \max\left\{ \sum_{j=1}^{n+1} |\beta_j| \ : \ \sum_{j=1}^{n+1} \beta_j x^{(j)} \in \Omega, \ \sum_{j=1}^{n+1} \beta_j = 1 \right\} = \max\left\{ \sum_{j=1}^{n+1} |\beta_j| \ : \ \sum_{j=1}^{n+1} \beta_j y^{(j)} \in \Omega', \ \sum_{j=1}^{n+1} \beta_j = 1 \right\} = \|P'\|_{\Omega'}.$$

Теорема доказана. □

**Следствие 1.** *Пусть $S$ — невырожденный $n$-мерный симплекс, $E$ — его минимальный эллипсоид, $S'$ — правильный симплекс, вписанный в единичный шар $B_n$. Предположим, что $P : C(E) \longrightarrow \Pi_1(\mathbb{R}^n)$ и $P' : C(B_n) \longrightarrow \Pi_1(\mathbb{R}^n)$ — интерполяционные проекторы, узлы которых совпадают соответственно с вершинами $S$ и $S'$. Тогда $\|P\|_E = \|P'\|_{B_n}$.*

*Доказательство.* Рассмотрим невырожденное аффинное отображение, которое переводит симплекс $S$ в правильный симплекс $S'$. Это отображение переводит эллипсоид $E$ в шар $B_n$. Остаётся применить теорему 3 в ситуации, когда $\Omega$ есть минимальный эллипсоид симплекса $S$. □

Дополним следствие 1 замечанием, справедливость которого устанавливается по той же схеме. Обозначим здесь через $\lambda_j$ базисные многочлены Лагранжа произвольного симплекса $S$. Точки минимального эллипсоида $E$, в которых достигается максимум функции $\lambda(x) = \sum |\lambda_j(x)|$, имеют то же геометрическое описание, которое выражается теоремой 1. В условии теоремы правильный симплекс нужно заменить произвольным, а описанный шар — минимальным эллипсоидом симплекса. В указанных точках границы эллипсоида значение $\lambda(x)$ максимально и равно $\|P\|_E$.

**Следствие 2.** *Существует универсальная константа $C > 0$, такая что для любого эллипсоида $E \subset \mathbb{R}^n$ и любого интерполяционного проектора, узлы которого принадлежат $E$, выполняются неравенства $\|P\|_E \geq \chi_n^{-1}(K_n) > C\sqrt{n}$.*

Сразу получается из (11) и предыдущего следствия.

## 3. Об одном экстремальном свойстве правильного симплекса, вписанного в шар

Пусть $S$ — невырожденный $n$-мерный симплекс, $E$ — минимальный эллипсоид, содержащий $S$. Зафиксируем натуральное $m \leq \frac{n}{2}$. Каждому набору из $m$ вершин симплекса поставим в соответствие точку $y \in E$, определяемую следующим образом. Пусть $g$ есть центр тяжести $(m-1)$-мерной грани $S$, содержащей выделенные вершины, $h$ — центр тяжести $(n-m)$-мерной грани, содержащей остальные

$n+1-m$ вершин. Тогда $y$ есть точка пересечения прямой $(gh)$ с границей эллипсоида в направлении от $g$ к $h$.

Сформулируем следующую гипотезу.

(H1) *Для данного натурального* $m \le \frac{n}{2}$ *и любого невырожденного симплекса* $S \subset B_n$ *найдётся такой набор из* $m$ *вершин симплекса, для которого* $y \in B_n$.

Более сильный вариант гипотезы утверждает, что указанное свойство выполняется *для любого натурального* $m \le \frac{n}{2}$ (H2). Для наших целей достаточно, чтобы (H1) было справедливо для $m = k(n)$. Число $k = k(n)$ определяется в пункте 1.

**Теорема 4.** *Для* $m = 1$ *гипотеза (H1) верна.*

*Доказательство.* Пусть $S$ — симплекс с вершинами $x^{(j)} \in B_n$ и центром тяжести $c$. Центр минимального для $S$ эллипсоида также находится в точке $c$. Поэтому в случае $m = 1$ точки $y$ имеют вид $y^{(j)} = 2c - x^{(j)}$, $j = 1, \dots, n+1$. Тем самым требуется показать, что для некоторой вершины $x$ симплекса верно $\|2c - x\| \le 1$. Так как $S$ является невырожденным, найдётся вершина $x$, удовлетворяющая неравенству $(c, x - c) \ge 0$. Для этой вершины

$$\|2c - x\|^2 = (2c - x, 2c - x) = 4(c, c - x) + \|x\|^2 \le \|x\|^2 \le 1,$$

Значит, вершина $x$ является подходящей. Теорема доказана. □

Обозначим через $\theta_n(B_n)$ минимальную норму интерполяционного проектора $P : C(B_n) \longrightarrow \Pi_1(\mathbb{R}^n)$, узлы которого принадлежат $B_n$. Через $P'$ обозначим проектор, узлы которого совпадают с вершинами правильного симплекса $S'$, вписанного в $B_n$.

**Теорема 5.** *Пусть* $n$ *таково, что утверждение (H1) верно для* $m = k(n)$. *Тогда* $\theta_n(B_n) = \|P'\|_{B_n}$.

*Доказательство.* Рассмотрим произвольный интерполяционный проектор $P$ с узлами $x^{(j)} \in B_n$. Пусть $S$ — симплекс с вершинами в этих точках, $\lambda_j$ — базисные многочлены Лагранжа для $S$. Обозначим через $E$ минимальный эллипсоид этого симплекса. Поскольку $S \subset B_n$, для некоторого набора из $k = k(n)$ вершин симплекса соответствующая точка $y$ принадлежит шару. Зафиксируем $y$ и запишем следующую цепочку соотношений:

$$\|P'\|_{B_n} = \|P\|_E = \max_{x \in E} \sum_{j=1}^{n+1} |\lambda_j(x)| = \sum_{j=1}^{n+1} |\lambda_j(y)| \le \max_{x \in B_n} \sum_{j=1}^{n+1} |\lambda_j(x)| = \|P\|_{B_n}.$$

Мы применили формулу для нормы проектора, теорему 1, следствие 1, а также замечание после него. Неравенство в этой цепочке следует из того, что $y \in B_n$. Заметим, что если $y$ является внутренней точкой шара, то это неравенство является строгим.

Таким образом, для любого проектора, узлы котрого принадлежат $B_n$, выполняется неравенство $\|P'\|_{B_n} \le \|P\|_{B_n}$. Следовательно, $\theta_n(B_n) = \|P'\|_{B_n}$, что и требовалось доказать. □

**Следствие 3.** *Если* $1 \le n \le 4$, *то* $\theta_n(B_n) = \|P'\|_{B_n}$.

*Доказательство.* В случае $n = 1$ утверждение эквивалентно тому, что минимальная норма интерполяционного проектора $P : C[-1, 1] \longrightarrow \Pi_1(\mathbb{R})$ реализуется для проектора, узлы которого совпадают с концами отрезка $[-1, 1]$. Если же $2 \le n \le 4$, то выполняется равенство $k(n) = 1$ и нужный результат сразу следует из теорем 4–5. □

Результат следствия 3 был установлен в [2] другим методом, подходящим лишь для размерностей со свойством $k(n) = 1$. Начиная с $n = 5$ выполняется неравенство $k(n) > 1$ (см. [2]). Однако равенство $\theta_n(B_n) = \|P'\|_{B_n}$ может быть получено на пути, отмеченном теоремой 5.

В утверждениях этого пункта единичный шар $B_n$ может быть заменён на произвольный евклидов шар $B$, что приведёт к эквивалентным результатам.

## References


[1] M. V. Nevskii, *Geometricheskie Ocenki v Polinomial'noj Interpolyacii*. Yaroslavl: P. G. Demidov Yaroslavl State University, 2012, in Russian.

[2] M. V. Nevskii and A. Y. Ukhalov, "Linear Interpolation on a Euclidean Ball in $\mathbb{R}^n$", *Modeling and Analysis of Information Systems*, vol. 26, no. 2, pp. 279–296, 2019. DOI: 10.18255/1818-1015-2019-2-279-296.

[3] M. V. Nevskii and A. Y. Ukhalov, "On Optimal Interpolation by Linear Functions on an $n$-Dimensional Cube", *Modeling and Analysis of Information Systems*, vol. 25, no. 3, pp. 291–311, 2018. DOI: 10.18255/1818-1015-2018-3-291-311.

[4] M. Nevskii and A. Ukhalov, "Perfect Simplices in $\mathbb{R}^5$", *Beitr. Algebra Geom.*, vol. 59, no. 3, pp. 501–521, 2018. DOI: 10.1007/s13366-018-0386-6.

[5] M. V. Nevskii, "On Some Problems for a Simplex and a Ball in $\mathbb{R}^n$", *Modeling and Analysis of Information Systems*, vol. 25, no. 6, pp. 680–691, 2018. DOI: 10.18255/1818-1015-2018-6-680-691.

[6] M. V. Nevskii, "Geometric Estimates in Interpolation on an n-Dimensional Ball", *Modeling and Analysis of Information Systems*, vol. 26, no. 3, pp. 441–449, 2019. DOI: 10.18255/1818-1015-2019-3-441-449.

[7] M. V. Nevskii, "Computation of the Longest Segment of a Given Direction in a Simplex", *Journal of Mathematical Sciences*, vol. 203, no. 6, pp. 851–854, 2014.

[8] F. John, "Extremum Problems with Inequalities as Subsidiary Conditions", in *Studies and essays presented to R. Courant on his 60th birthday (Jan. 8, 1948)*, New York: Interscience, 1948, pp. 187–204.

[9] K. Ball, *Ellipsoids of Maximal Volume in Convex Bodies*, arXiv: math/9201217v1 [math. MG], Sep 25, 1990.

[10] K. Ball, "An Elementary Introduction to Modern Convex Geometry", *Math. Sci. Res. Inst. Publ.*, vol. 31, no. 1, pp. 1–58, 1997.

[11] L. Fejes Tót, *Regular Figures*. New York: Macmillan/Pergamon, 1964.

[12] D. Slepian, "The Content of Some Extreme Simplices", *Pacific J. Math*, vol. 31, pp. 795–808, 1969.

[13] D. Vandev, "A Minimal Volume Ellipsoid around a Simplex", *C. R. Acad. Bulg. Sci.*, vol. 45, no. 6, pp. 37–40, 1992.

[14] G. M. Fikhtengol'ts, *Kurs Differencial'nogo i Integral'nogo Ischisleniya. Tom 3*. Moscow: Fizmatlit, 2001, in Russian.

[15] A. P. Prudnikov, Y. A. Brychkov, and O. I. Marichev, *Integraly i Ryady*. Moscow: Nauka, 2002, in Russian.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 28, NO. 2, 2021
journal homepage: www.mais-journal.ru

THEORY OF COMPUTING

# The System for Transforming the Code of Dataflow Programs into Imperative

V. S. Vasilev[1], A. I. Legalov[2], S. V. Zykov[2]

DOI: 10.18255/1818-1015-2021-2-198-214

[1]Siberian Federal University, 79 Svobodny Ave., Krasnoyarsk 660041, Russia.
[2]Higher school of Economics, National research University, 20 Myasnitskaya str., Moscow 101000, Russia.

MSC2020: 68N15, 68Q10
Research article
Full text in Russian

Received May 7, 2021
After revision May 31, 2021
Accepted June 2, 2021

Functional dataflow programming languages are designed to create parallel portable programs. The source code of such programs is translated into a set of graphs that reflect information and control dependencies. The main way of their execution is interpretation, which does not allow to perform calculations efficiently on real parallel computing systems and leads to poor performance. To run programs directly on existing computing systems, you need to use specific optimization and transformation methods that take into account the features of both the programming language and the architecture of the system. Currently, the most common is the Von Neumann architecture, however, parallel programming for it in most cases is carried out using imperative languages with a static type system. For different architectures of parallel computing systems, there are various approaches to writing parallel programs. The transformation of dataflow parallel programs into imperative programs allows to form a framework of imperative code fragments that directly display sequential calculations. In the future, this framework can be adapted to a specific parallel architecture. The paper considers an approach to performing this type of transformation, which consists in allocating fragments of dataflow parallel programs as templates, which are subsequently replaced by equivalent fragments of imperative languages. The proposed transformation methods allow generating program code, to which various optimizing transformations can be applied in the future, including parallelization taking into account the target architecture.

**Keywords:** transformation of programs; dataflow parallel programming; program analysis; typing; intermediate program representations

INFORMATION ABOUT THE AUTHORS

| | |
|---|---|
| Vladimir S. Vasilev<br>correspondence author | orcid.org/0000-0002-3340-6678. E-mail: vsvasilev@sfu-kras.ru<br>Senior lecturer. |
| Alexander I. Legalov | orcid.org/0000-0002-5487-0699. E-mail: alegalov@hse.ru<br>Doctor of Science, Professor. |
| Sergey V. Zykov | orcid.org/0000-0002-2115-5461. E-mail: szykov@hse.ru<br>Doctor of Science, Professor. |

**For citation**: V. S. Vasilev, A. I. Legalov, and S. V. Zykov, "The System for Transforming the Code of Dataflow Programs into Imperative", *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 198-214, 2021.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 28, № 2, 2021
сайт журнала: www.mais-journal.ru

THEORY OF COMPUTING

# Трансформация функционально-потоковых параллельных программ в императивные

В. С. Васильев[1], А. И. Легалов[2], С. В. Зыков[2]

DOI: 10.18255/1818-1015-2021-2-198-214

[1]Сибирский федеральный университет, пр. Свободный, д. 82, г. Красноярск, 660041 Россия.
[2]Национальный исследовательский университет «Высшая школа экономики», ул. Мясницкая, д. 20, г. Москва, 101000 Россия.

УДК 004.4'42
Научная статья
Полный текст на русском языке

Получена 7 мая 2021 г.
После доработки 31 мая 2021 г.
Принята к публикации 2 июня 2021 г.

Функционально-потоковая парадигма параллельного программирования ориентирована на разработку параллельных переносимых программ. Исходный код функционально-потоковых программ транслируется в набор графов, отражающих информационные и управляющие зависимости. Основным способом их исполнения является интерпретация, что не позволяет эффективно выполнять вычисления на реальных параллельных вычислительных системах и ведет к низкой производительности. Для непосредственного выполнения программ на существующих вычислительных системах требуется использование специфических методов оптимизации и трансформации, учитывающих особенности как языка программирования, так и архитектуры исполнителя. В настоящее время наиболее распространенной является архитектура Фон-Неймана, параллельное программирование для которой в большинстве случаев осуществляется с использованием языков, поддерживающих императивный стиль и ориентированных на статическую систему типов. Для различных архитектур параллельных вычислительных систем существуют разнообразные подходы к написанию параллельных программ. Трансформация функционально-потоковых параллельных программ в императивные позволяет сформировать общий каркас из фрагментов императивного кода, непосредственно отображающих последовательные вычисления, который в дальнейшем может быть адаптирован к конкретной параллельной архитектуре. В работе рассматривается подход к выполнению такого типа трансформации, заключающийся в выделении фрагментов функционально-потоковых параллельных программ в качестве шаблонов, заменяемых впоследствии на эквивалентные фрагменты императивных языков. Предлагаемые методы трансформации позволяют порождать программный код, к которому в дальнейшем можно применять различные оптимизирующие преобразования, включая распараллеливание с учетом целевой архитектуры.

**Ключевые слова:** трансформация программ; функционально-потоковое параллельное программирование; анализ программ; типизация; промежуточные представления программ

## ИНФОРМАЦИЯ ОБ АВТОРАХ

Владимир Сергеевич Васильев
автор для корреспонденции
orcid.org/0000-0002-3340-6678. E-mail: vsvasilev@sfu-kras.ru
старший преподаватель.

Александр Иванович Легалов
orcid.org/0000-0002-5487-0699. E-mail: alegalov@hse.ru
доктор технических наук, профессор.

Сергей Викторович Зыков
orcid.org/0000-0002-2115-5461. E-mail: szykov@hse.ru
доктор технических наук, профессор.

**Для цитирования**: V. S. Vasilev, A. I. Legalov, and S. V. Zykov, "The System for Transforming the Code of Dataflow Programs into Imperative", *Modeling and analysis of information systems*, vol. 28, no. 2, pp. 198-214, 2021.

## Введение

Основным направлением в параллельном программировании является написание кода, ориентированного на целевую архитектуру независимо от того, насколько эффективно система программирования отражает специфику предметной области. В большинстве случаев это связано со стремлением получить эффективный код в ущерб эффективности и надежности процесса разработки. Постоянное совершенствование и изменение архитектур параллельных вычислительных систем (ПВС) ведет к тому, что ранее написанные параллельные программы приходится перерабатывать, зачастую достаточно сильно, для адаптации к новым условиям эксплуатации. За относительно короткий срок сменилось несколько разновидностей архитектур высокопроизводительных систем. В настоящее время встречаются различные комбинированные решения, в которых, зачастую одновременно, используются механизм передачи сообщений, многопоточность, графические ускорители, решения на уровне систем на кристалле [1].

Разнообразие ПВС ведет к поиску подходов, которые могли бы поддержать архитектурно-независимую разработку параллельных программ и обеспечить их эффективную трансформацию нужную в целевую архитектуру. Можно выделить ряд таких подходов, ориентированных на исходное создание программ для абстрактных параллельных вычислителей и последующую трансформацию в реальные архитектуры. Концепция ресурсно-независимого параллельного программирования реализована в языке COLAMO, ориентированного на разработку систем на кристалле [2, 3]. Создание универсальных языков, напрямую не связанных с архитектурными ограничениями, можно проследить на примере функциональных языков параллельного программирования Sisal [4] и Пифагор [5]. В языке Set@l программа представляет собой описания алгоритма в виде архитектурно-независимого информационного и набора архитектурно-зависимых аспектов, что позволяет переносить код между различными параллельными архитектурами без изменения алгоритма [6].

В языке функционально-потокового параллельного программирования Пифагор реализована концепция единственного использования вычислительных ресурсов, что позволяет формировать параллельные программы, ориентированные на архитектурную независимость. Для анализа возможностей языка разработаны инструментальные средства, поддерживающие процесс создания, преобразования и выполнения функционально-потоковых параллельных программ [7]. Показано, что написанные программы могут быть оптимизированы [8], отлажены [9] и использованы для формальной верификации [10] программ еще до того, как они будут выполняться на конкретной вычислительной системе.

Несмотря на многоэтапный процесс компиляции и формирование промежуточных представлений, позволяющих порождать выходное представление на любом императивном языке, на текущий момент была реализована только интерпретация, базирующаяся на этих промежуточных представлениях. В связи с этим актуальной является задача трансформации программы в выходные императивные представления, что позволяет выполнять код, после дополнительных компиляций, на современных вычислительных архитектурах без дополнительных накладных расходов. К таким императивным языкам относятся, например, C, C++, Фортран.

Трансформация функционально-потоковых параллельных (ФПП) программ в языки со статической типизацией затруднена из-за того, что в языке Пифагор используется динамическая типизация данных. Однако можно отметить, что существует ряд общих принципов, связанных с процессом трансформации, которые напрямую не связаны с системой типов и могут быть реализованы независимо от нее. Можно также отметить императивные языки, в которых реализована динамическая типизация данных, например, Python. И хотя Python не используется в высокопроизводительных параллельных вычислениях, существуют предметные области, в которых на нем разрабатываются многопоточные параллельные программы. Поэтому трансформация функционально-потоковых параллельных программ в императивные может представлять интерес и для таких языков.

Вместе с тем для функциональных языков программирования, к которым относится и язык Пифагор, проработана концепция вывода типов [11] из поступающих на вход аргументов, что, в принципе, позволяет применить данную концепцию для трансформации ФПП программ. Наряду с этим также можно отметить, что на базе функционально-потоковой парадигмы параллельного программирования предложен статически типизированный язык Smile [12, 13], при разработке компилятора которого планируется использовать рассматриваемые решения. Другим вариантом использования, также связанного с ФПП программированием, является введение явной типизации в реверсивный информационный граф, формируемый в результате компиляции программ, написанных на Пифагоре. Этот подход предлагается в работе.

## 1. Особенности трансформации функционально-потоковых параллельных программ

К особенностям языка ФПП программирования Пифагор, определяющим специфику трансформации в императивный код следует отнести:

1) Использование динамической типизации, при которой одна и та же функция может получать в качестве аргумента данные различного типа, что порождает результат, тип которого также может быт различным.
2) В языке имеются специфические операторы, отсутствующие в императивных языках, например: задержанный список, параллельный список. Они используются специфическим образом, что предопределяет особый подход к их анализу и трансформации.
3) В ходе выполнения, вычисления некоторых ветвей алгоритма могут приводить к ошибкам, которые не влияют на работу алгоритма если их результаты не используются.
4) Используется управление вычислениями по готовности данных.

В связи с этим трансформация исходных программ в статически типизированные программы для повышения эффективности связана с наложением определенных ограничений.

Трансформация программ осуществляется на основе анализа реверсивного информационного графа (РИГ), являющего результатом компиляции исходных текстов программы на языке Пифагор [7]. Данный граф сохраняет всю информацию и при этом отображает ее в форме, удобной для дальнейшего анализа. В ходе анализа можно выделить следующие основные фазы, ориентированные на анализ и трансформацию различных комбинаций программных объектов:

- списков данных, обеспечивающих структурирование;
- предопределенных функций общего вида;
- специализированных предопределенных функций;
- функций, разработанных пользователем;
- параллельных списков, определяющих массовые вычисления;
- задержанных списков, ориентированных на альтернативное выполнение операторов.

При анализе РИГ и формировании корректной последовательности операций императивной программы учитывается зависимость по данным. Помимо этого, для формирования различных конструкций императивной программы, узлы РИГ формируются в группы (шаблоны), образующие конструкции, необходимые для преобразования. Эти шаблоны в тексте поясняются соответствующими эквивалентными фрагментами кода на языке Пифагор. Порождаемые в ходе трансформации императивные конструкции иллюстрируются на языке программирования C++, который используется для формирования выходного представления.

## 2. Использование статической типизации для трансформации функционально-потоковых параллельных программ

Основной областью применения параллельного программирования являются высокопроизводительные вычисления. Применяемые при этом языки программирования являются статически

типизированными, что позволяет избавиться от динамической проверки типов данных во время вычислений и повышает производительность параллельных программ. Исходя из этого основной акцент в работе сделан на трансформацию в императивные языки, поддерживающие статическую типизацию. Однако следует отметить, что в языке ФПП программирования Пифагор используется динамическая типизация данных. В следующем варианте языка ФПП программирования Smile [12, 13] применяется статическая типизация, облегчающая процесс трансформации. Переход в данном языке к статической типизации ведет также к изменению аксиоматики языка. В большинстве случаев это связано с упрощением семантики операторов и предопределенных функций, что обуславливается стремлением повысить эффективность процесса трансформации. Однако для данного языка еще не разработан комплект инструментальных средств, обеспечивающих компиляцию исходных программ в промежуточное представление, позволяющее проводить дальнейший анализ и компиляцию. Поэтому для отработки методов трансформации принято решение использовать реверсивный информационный граф, порождаемый компилятором языка Пифагор с последующим явным добавлением к нему информации о типах данных. Подобные решения уже использовались при трансформации ФПП программ в программируемые логически интегральные схемы (ПЛИС) [14, 15], а также при разработке методов формальной верификации [16]. Планируется, что полученные результаты будут учтены при генерации промежуточного представления компилятором языка Smile. Для этого анализу и дальнейшему преобразованию в императивный код подвергаются не все конструкции языка Пифагор. В ряде случаев на преобразуемые конструкции накладываются дополнительные ограничения, что позволяет выстроить более эффективные методы трансформации.

Тип результата для каждого узла РИГ, определяющего оператор, может быть вычислен если известен тип входного аргумента функции, а также типы результатов, возвращаемых всеми используемыми внешними функциями. Исходя из этого, сделав соответствующую начальную разметку этих узлов, можно полностью сформировать статически типизированное представление РИГ. Результатом работы компилятора ФПП программ, является набор РИГ. Для добавления статической типизации предложено ввести дополнительное описание, связанное с каждой вершиной, в следующем формате:

```
<Тип функции> ::= <Тип аргумента> -> <Тип возвращаемого значения>.
<Тип кортежа> ::= @{<Элементы кортежа>}
<Элементы кортежа> ::= <пусто> | <Тип> | <Тип> [,<Элементы кортежа>]
<Тип> ::= @char | @int | @double | <Тип списка> | <Тип функции>
<Тип списка> ::= @(<Тип>) | @[<Тип>]
<Тип возвращаемого значения> ::= <Тип>
```

Функция на языке Пифагор всегда имеет один аргумент, который, однако, может задаваться списком (кортежем), содержащим несколько элементов различных типов.

Семантика модели вычислений гарантирует, что на вход не будет подан параллельный список. Он может быть сформирован в результате выполнения функции. Поэтому в описании возможно как задание списка данных *@(<Tun>)*, так и параллельного списка *@[<Tun>]*. Описание типов сопоставляется с каждым узлом РИГ и размещается в отдельном файле, что позволяет использовать немодифицированный РИГ в ранее разработанных инструментальных средствах. Данные в дополнительный файл в настоящий момент вносятся вручную. Однако промежуточные типы, как и в других языках функционального программирования [11], могут автоматически выводиться из информации об аргументах текущей функции, а также известных сигнатур функций, используемых в РИГ.

**Table 1.** Example of converting binary and unary arithmetic operators

**Таблица 1.** Пример преобразования бинарных и унарных арифметических операторов

| Фрагмент кода на Пифагор | Эквивалентный фрагмент на C++ |
|---|---|
| `// @{@int} -> @int`<br>`fact << funcdef X {`<br>`  X1 << (X, 1);`<br>`  [(X1:[<=,>]):?]^(`<br>`  { X },`<br>`  { (X, X1:-:fact ):*  }`<br>`  ):. >> return;`<br>`}` | `int fact (int arg_0 ){`<br>`  int var1 = 1;`<br>`  int var2 = 1;`<br>`  int var3;`<br>`  if (arg_0 <= var1){`<br>`  var3 = arg_0;`<br>`  }`<br>`  if (arg_0 > var1){`<br>`  int var4 = arg_0 - var2;`<br>`  int var5 = fact(var4);`<br>`  int var6 = arg_0 * var5;`<br>`  var3 = var6;`<br>`  }`<br>`  return var3;`<br>`}` |

В таблице 1 приведена функция вычисления факториала на языке Пифагор и результат ее преобразования в императивную программу на C++. В первой строке исходного кода на Пифагор приведено описание типа аргумента функции, на основе которого выведены остальные типы.

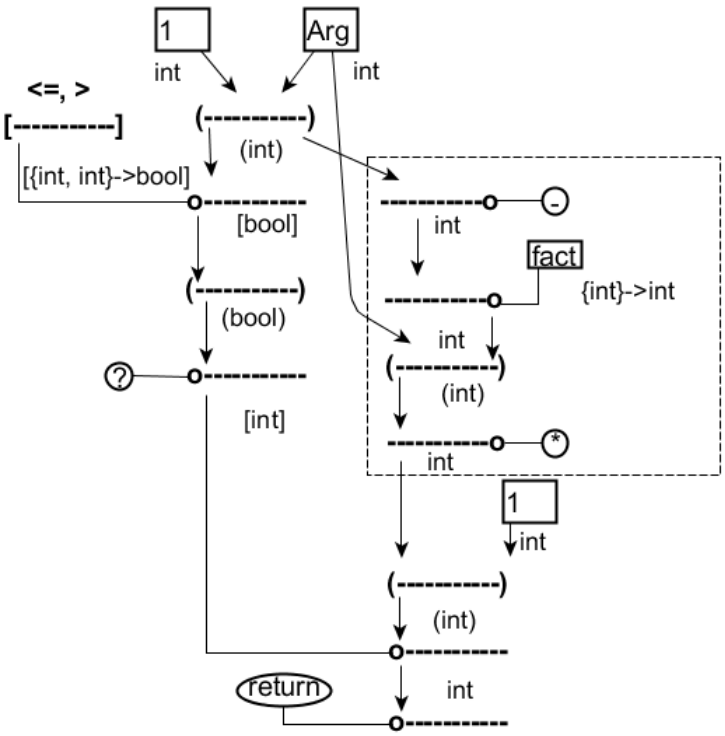



**Fig. 1.** Example of marking up a reverse information graph with types

**Рис. 1.** Пример разметки реверсивного информационного графа типами

В таблице 2 приведено текстовое представление РИГ, генерируемое системой трансляции. Показанные типы выставляются при первичной разметке графа, затем они пересчитываются для ряда узлов с учетом правил эквивалентных преобразований [16]. Например, узел 11 – группировка в параллельный список – добавлен в результате трансляции задержанного списка, так как семантика модели вычислений допускает описания в задержанном списке параллельных вычислений. Однако так как задержанный список в данном случае используется лишь для организации ветвления, то в ходе преобразования в императивную форму он распознается как «шаблон», и узел 11 устраняется. Кроме того, узел 6 в программе используется для задания необходимых приоритетов выполнения операций – перед преобразованием в императивную форму такие узлы удаляются.

**Table 2.** Example of marking up a reverse information graph with types

**Таблица 2.** Пример разметки реверсивного информационного графа типами

| Текстовое представление РИГ | Тип данных узла |
|---|---|
| External<br>0 fact<br>Local<br>0 1<br>1 {2}11<br>id delay operation links<br>0 0 arg<br>1 0 (---) 0 loc:0<br>2 0 [---] <= ><br>3 0 : 1 2<br>4 0 (---) 3<br>5 0 : 4 ?<br>6 0 [---] 5<br>7 2 : 1 -<br>8 2 : 7 ext:0<br>9 2 (---) 0 8<br>10 2 : 9 *<br>11 2 [---] 10<br>12 0 (---) loc:0 loc:1<br>13 0 : 12 6<br>14 0 : 13 .<br>15 0 return 14 | ------------------<br>{int} -> int<br>------------------<br>int<br>{[int]}<br>------------------<br>int<br>(int)<br>[{int, int} -> bool]<br>[bool]<br>(bool)<br>int<br>[int]<br>int<br>int<br>(int)<br>int<br>[int]<br>(int)<br>int<br>int<br>int |

На рисунке 1 размеченный РИГ приведен в сокращенном графическом формате (задержанные списки показаны рамкой), при этом убраны узлы, не участвующие в процессе трансформации.

## 3. Особенности трансформации различных конструкций языка ФПП программирования

Операторы языка ФПП программирования образуют различные комбинации, которые могут рассматриваться при трансформации как группы, определяющие различные шаблоны. Каждый из этих шаблонов может порождать на выходе различные конструкции императивных языков программирования. Ниже рассмотрены типичные ситуации, связанные с проводимыми трансформациями.

### 3.1. Трансформация списков данных

В языке с динамической типизацией списки данных в общем случае представляют иерархически вложенные коллекции, содержащие данные различного типа. Они также играют различные роли как при использовании в качестве аргументов функций, так и внутри функций. На данный вид списков наложены следующие ограничения.

1) Списки данных могут использоваться для формирования только одномерных массивов, элементы которых принадлежат к одному типу. Учитывая то, что в языке отсутствует описание типов, элементы списка могут иметь только предопределенный (базовый) тип. К базовым типам относятся целые и действительные числа, булевские величины, символы. Для таких ограниченных списков задается описание размера, что позволяет рассматривать их как одномерные массивы.
2) Список данных может являться кортежем, состоящим из данных разного типа. В таком виде он может использоваться для задания описаний аргументов функций, а также в качестве списка фактических параметров, поступающих на вход вызываемых функций. Тип элементов в данном случае также ограничен имеющимися предопределенными типами данных или указателями на предопределенные типы. Указатели позволяют передавать массивы.

### 3.2. Предопределенные функции

В функционально-потоковой модели параллельных вычислений (ФПМПВ) определен ряд функций, которые не имеют аналогов в императивных языках, но используются для организации вычислений. Их трансформация связана с заменой на специально написанные императивные функции. Следует также отметить, что не все функции данной модели поддаются эффективной трансформации. Поэтому их преобразование не имеет смысла. Подобный подход также принят при создании статически типизированного языка ФПП программирования Smile, в котором ряд функций, используемых в языке Пифагор, отсутствует. При оставшихся функций могут вноситься ограничения по семантике, что связано с их изначально универсальностью, которая не всегда является необходимой в статически типизированных программах.

Существует набор предопределенных функций, которые при наличии дополнительной разметки типов, сопоставляемой вершинам РИГ, достаточно просто трансформируются в операции императивного языка программирования. В частности, к ним относятся арифметические и логические функции, функции сравнения, ряд вспомогательных функций, манипулирующих контейнерными типами и атомарными данными.

**Функция** «|» возвращает длину списка данных. Поэтому его легко сопоставить с функциями, осуществляющими вычисление длины. В стандартной библиотеке многих императивных языков имеются структуры данных, поддерживающие соответствующую операцию. Например, в C++ это могут быть контейнеры типа vector или list.

**Арифметические и логические функции, функции сравнения** могут быть непосредственно сопоставлены с соответствующими операциями. В ходе трансформации нужно учесть, что для ряда операторов существуют унарные и бинарные формы. Кроме того, такие операторы языка Пифагор могут обрабатывать значения различных типов. Решение задачи сопоставления операций

осуществляется на основе анализа типов данных. В таблице 3 приведен пример выполнения подобных преобразований над функциями определения модуля числа и сравнения двух дробных чисел с погрешностью.

**Table 3.** Example of converting binary and unary arithmetic operators

**Таблица 3.** Пример преобразования бинарных и унарных арифметических операторов

Фрагмент кода на Пифагор

```
//@{@double}->@double
d_abs << funcdef X {
  [((X, 0.0):[<, >=]):?]^(
    {X:-},
    {X}
  ):. >> return;
}

//@{@double,@double,@double}->@bool
is_close << funcdef X {
  A << X:1;
  B << X:2;
  Eps << X:3;
  return << ((A,B):-:d_abs,Eps):<;
}
```

Эквивалентный фрагмент на C++

```
double d_abs (double arg_0 ) {
  double var1 = 0.0;
  double var2;
  if (arg_0 < var1) {
    double var3 = -arg_0;
    var2 = var3;
  }
  if (arg_0 >= var1) {
    var2 = arg_0;
  }
  return var2;
}

double is_close (double arg_0,
                 double arg_1 ,double arg_2 ) {
  double var1 = arg_0 - arg_1;
  double var2 = d_abs(var1);
  bool var3 = var2 < arg_2;
  return var3;
}
```

В ряде ситуаций предопределенные функции языка программирования Пифагор используются в типовых комбинациях, которые можно рассматривать совместно. Неэффективная реализация

отдельных функций в этом случае может быть заменена на более эффективное решение в формируемом на языке C++ выходном представлении. Выделение фрагментов, образующих шаблоны, ведется на основе анализа реверсивного информационного графа. В частности это касается функций транспонирования и формирования дубликатов.

**Функция транспонирования «#»** выполняет преобразование для вложенных списков данных, аналогичное транспонированию матриц и чаще всего используется для построения пар аргументов, которые в дальнейшем вычисляются параллельно одной и той же функцией. Примером может служить попарное сложение элементов двух векторов. В сочетании с формированием на выходе параллельного списка данная функция производит одновременное сложение всех пар элементов векторов A и B, получая на выходе вектор C. Анализируя данные фрагменты кода можно заменить их на более эффективные императивные решения, которые вместо транспонирования используют прямое попарное сложение элементов массивов. Результат такой трансформации приведен в таблице 4.

**Table 4.** Transformation of a template fragment containing a predefined transpose function

**Таблица 4.** Трансформация шаблонного фрагмента, содержащего предопределенную функцию транспонирования

Фрагмент кода на Пифагор

```
//@{@(@int),@(@double)}->@(@double)
vec_sum << funcdef vecPair {
  A << vecPair:1;
  B << vecPair:2;
  C << (A,B):#:[]:+; // Трансформируемый фрагмент
  return << C;
}
```

Эквивалентный фрагмент на C++

```
vector<double > vec_sum (vector<double > arg_0 ,vector<double > arg_1 ){
  size_t var1_size = arg_0.size();
  vector<double > var1(var1_size);
  for (size_t var1_iter = 0; var1_iter < var1_size; ++var1_iter){
    var1[var1_iter] = arg_0[var1_iter] + arg_1[var1_iter];
  }
  return var1;
}
```

**Функция «*dup*»** выполняет дублирование данных и используется чаще всего для организации параллельных вычислений, когда один из аргументов остается неизменным. Данное дублирование в последовательном императивном программировании является избыточным, так как легко заменяется на повторное обращение к одной и той же переменной. Поэтому непосредственная реализация функции вряд ли имеет смысл. Целесообразнее на основе анализа более крупных фрагментов рассматривать эту функцию как часть общих преобразований, в которых происходит многократное обращение к одной и той же ячейке памяти.

**Функция «..»** является генератором последовательности данных, образующих ряд целых или действительных чисел с заданным шагом. Для его порождения используется задание начала интервала, его конца и шаг. На выходе формируется параллельный список чисел в заданном диапазоне и с заданным шагом. Результаты работы генератора чаще всего используются в циклах как значение

счетчика. Данную функцию также можно рассматривать в составе соответствующих шаблонов, что позволяет интегрировать ее в операторы цикла императивных программ. Пример выделения такого шаблона и порождения на его основе фрагмента императивного кода представлен в таблице 5.

**Table 5.** Converting number sequence generators to imperative form

**Таблица 5.** Преобразование генераторов последовательности чисел в императивную форму

Фрагмент кода на Пифагор

```
//@{@(@double)}->@(@double)
ex18 << funcdef X {
  Len << X:|;
  Generator << (2,Len,3):..;
  Sub << X:Generator;
  return << Sub;
}

//@{@(@double)}->@(@double)
ex19 << funcdef X {
  len << X:|;
  return << (X,(1,len):..):#:[]:ex5;
}
```

Эквивалентный фрагмент на C++

```
vector<double > ex18 (vector<double > arg_0 ){
  int var1 = 2;
  int var2 = 3;
  int var3 = arg_0.size();
  size_t var4_size = (var3 - var1)/var2;
  vector<double > var4(var4_size);
  for (int var4_iter_in = 0, var4_iter_ex = var1;
       var4_iter_ex < var3;
       ++var4_iter_in, var4_iter_ex += var2) {
    var4[var4_iter_in] = arg_0[var4_iter_ex];
  }
  return var4;
}

vector<double > ex19 (vector<double > arg_0 ){
  int var1 = 1;
  size_t var2_size = arg_0.size();
  vector<double> var2(var2_size);
  for (size_t var2_iter = 0; var2_iter < var2_size; ++var2_iter){
    var2[var2_iter] = ex5(arg_0[var2_iter], [var2_iter]);
  }
  return var2;
}
```

### 3.3. Задержанные списки

Операторы, вложенные внутрь задержанного списка, не начинают выполняться даже при готовности всех своих аргументов до тех пор, пока список не будет «раскрыт». Задержанные списки в функционально-потоковом программировании используются, в том числе для организации ветвления, при этом:

- фрагменты кода, выполняемые условно, помещаются в различные задержанные списки, которые, в свою очередь, вкладываются в список данных;
- в зависимости от результата выполнения «условия» из списка данных выбирается один из элементов — задержанный список;
- к выбранному задержанному списку применяется оператор интерпретации, выполняющий «раскрытие списка», приводящее к выполнению всех вложенных операторов.

При преобразовании в императивную форму, операторы разных задержанных списков, представляющих собой альтернативные ветви алгоритма, списка необходимо разместить в разных блоках оператора ветвления, следовательно система преобразования должна учитывать не только зависимости по данным между узлами, но и вложенность узлов в задержанные списки. Преобразование задержанных списков, задающих ветвление, в императивную форму представлено в таблице 3 на примере трансформации функции **d_abs**.

### 3.4. Списки данных как аргументы функций

Списки данных являются одной из основных структур, используемых в ФПП программах. В текущей версии интерпретатора они обрабатываются как массивы [17]. Однако в ряде случаев списки данных используются как элементы синтаксических конструкций. При этом в императивном коде такие списки присутствовать не должны, так как преобразуются в другие программные объекты. Например, список, построенный из задержанных списков, в таблице 3 служит для организации ветвления.

Если функция принимает несколько аргументов, то перед вызовом они группируются в кортеж. В императивных языках аргументы целесообразнее передавать в виде списка параметров. Вместе с тем для обращения к конкретному аргументу функции в языке Пифагор выполняется обращение к элементу списка данных по индексу. Такие обращения при преобразовании должны заменяться на обращения к соответствующим аргументам функции, как показано в таблице 4.

### 3.5. Параллельные списки

Параллельные списки отличаются от списков данных тем, что применяемая к ним операция применяется к каждому вложенному элементу. Наряду со списками данных они зачастую служат вспомогательными нетрансформируемыми понятиями в составных конструкциях. РИГ, приведенный в таблице 1, содержит три задержанных списка: первый используется для задания приоритета операций, второй – для группировки операторов условия, третий был добавлен в результате трансляции задержанного списка. Ни один из них в явном виде не появится в императивном коде.

В связи с ориентацией на статическую типизацию предполагается, что параллельные списки анализируемой программы не имеют вложений и состоят только из элементов, имеющих одинаковый предопределенный тип (как и списки данных). Поэтому для их трансформации возможно использовать класс *vector* из стандартной библиотеки языка программирования C++.

Основное отличие процесса трансформации параллельных списков от списков данных заключается в контексте функций выполняемых над этим списком. На основе анализа этого контекста выделяется соответствующий шаблон, который помещает в цикл все функции, осуществляющие обработку всех элементов формируемого вектора. В ходе трансформации параллельные списки раскрываются, формируя циклические конструкции в соответствии с правилом ФПП-модели

вычислений:

$$[d_1, d_2, \ldots, d_n] : f \longrightarrow [d_1 : f, d_2 : f, \ldots, d_n : f].$$

В данной ситуации использование функции f с параллельным списком $[d_1, d_2, \ldots, d_n]$ приводит к ее применению к каждому элементу списка, то есть повторяющемуся действию. Однако параллельные списки зачастую описываются в программе неявно, а получаются в результате других вычислений. В примере преобразования, показанном в таблице 4, формируется параллельный список, состоящий из пар соответствующих элементов списков данных *vecPair:1* и *vecPair:2*.

### 3.6. Функции высшего порядка

Передача функции в качестве аргумента другой функции часто используется для повышения гибкости программ. Язык Пифагор имеет поддержку функций высшего порядка и они часто используются в программах. В качестве аргумента может передаваться не только функция, но также оператор и даже произвольный фрагмент кода, помещенный в задержанный список. Такие конструкции крайне сложно перенести на императивный язык в виду следующих причин:

- на вход арифметических и логических операторов могут быть поданы аргументы различных типов данных, но для преобразования необходимо указать типы явно;
- операторы **+** и **-** могут быть унарными;
- фрагмент кода, помещенный в задержанный список может иметь зависимости от контекста функции, в которой он был создан и содержать частично вычисленные значения.

В связи с этим, в программах, предназначенных для преобразования, не должны использоваться указанные выше синтаксические возможности ФПМПВ, а для передачи кода в качестве аргумента функции необходимо явно описывать вспомогательные функции – обертки и типизировать их, как показано в таблицах 6 и 2.

## 4. Инструментальная поддержка трансформации функционально-потоковых параллельных программ

В процессе преобразования осуществляется следующая трансформация типов данных ФПП программы:

- типы ***int, double*** и ***char*** заменяются на соответствующие типы языка C++;
- числовые списки данных и параллельные списки преобразуются в тип (***std::vector***), а списки символов в строки типа (***std::string***);
- в качестве аргументов функций высшего порядка используется тип ***std::function***.

В функционально-потоковом параллельном программировании используется принцип единственного присваивания, поэтому реверсивный информационный граф функции, по сути, задает граф программных зависимостей (ГПЗ) [18]. Такая форма используется как для внутреннего представления программы в системе преобразования.

Алгоритм преобразования применяется отдельно к каждой функции, при этом он сводится к тому, что выделяет в графе описанные выше шаблоны, для которых задан порядок преобразования. Реверсивный информационный граф обрабатывается в порядке распространения потока данных: от аргумента функции к оператору ***return***.

Выбранное внутреннее представление программы в системе преобразования использует ярусно-параллельную форму (ЯПФ), за счет этого отражает структуру ФПП-программы и позволет удобно и эффективно получать нужную для обработки информацию [19]. Преобразование выполняется, начиная с верхних ярусов ЯПФ, однако отдельно выделяются задержанные списки. В коде ФПП-программ такие списки чаще всего задают альтернативные блоки операторов. Несмотря на то, что узлы одного задержанного списка в ЯПФ располагаются на разных ярусах – в коде на императивном языке они размещаются последовательно друг за другом.

**Table 6.** Using auxiliary functions to pass operators as arguments to a function

**Таблица 6.** Использование вспомогательных функций для передачи операторов в качестве аргументов функции

Фрагмент кода на Пифагор

```
d_plus << funcdef X { //@{@double, @double}->@double
  return << (X:1, X:2):+;
}
//@{@(@double),@(@double), @{@double,@double}->@double}->@[@double]
map3_d_d << funcdef X {
  VectorA << X:1;
  VectorB << X:2;
  Operation << X:3;

  [((VectorA:|, VectorB:|):[!=, =]):?]^(
    {"badVectorSize":error},
    {(VectorA , VectorB):#:[]:Operation}
  ):. >> return;
}
// скалярное произведение векторов
//@{@(@double),@(@double)}->@(@double)
vec_mul_d_d << funcdef X {
  ((X:1, X:2, d_plus):map3_d_d) >> return;
}
```

Эквивалентный фрагмент на C++

```
double d_plus (double arg_0 ,double arg_1 ){
  double var1 = arg_0 + arg_1;
  return var1;
}
vector<double > map3_d_d (vector<double > arg_0 ,vector<double > arg_1 ,
function<double(double,double)> arg_2 ){
  int var1 = arg_0.size();
  int var2 = arg_1.size();
  vector<double > var3;
  if (var1 != var2){
    throw std::runtime_error("badVectorSize");
  }
  if (var1 == var2){
    size_t var4_size = arg_0.size();
    vector<double> var4(var4_size);
    for (size_t var4_iter = 0; var4_iter < var4_size; ++var4_iter){
      var4[var4_iter] = arg_2(arg_0[var4_iter], arg_1[var4_iter]);
    }
    var3 = var4;
  }
  return var3;
}
vector<double > vec_mul_d_d (vector<double > arg_0 ,vector<double > arg_1 ){
  vector<double > var1 = map3_d_d(arg_0, arg_1, d_plus);
  return var1;
}
```

На рисунке 2 окружностями обозначены узлы информационного графа, штриховой линией выделены задержанные списки, сверху в ромбе приведен номер задержанного списка, в восьмиугольниках записаны номера ярусов.

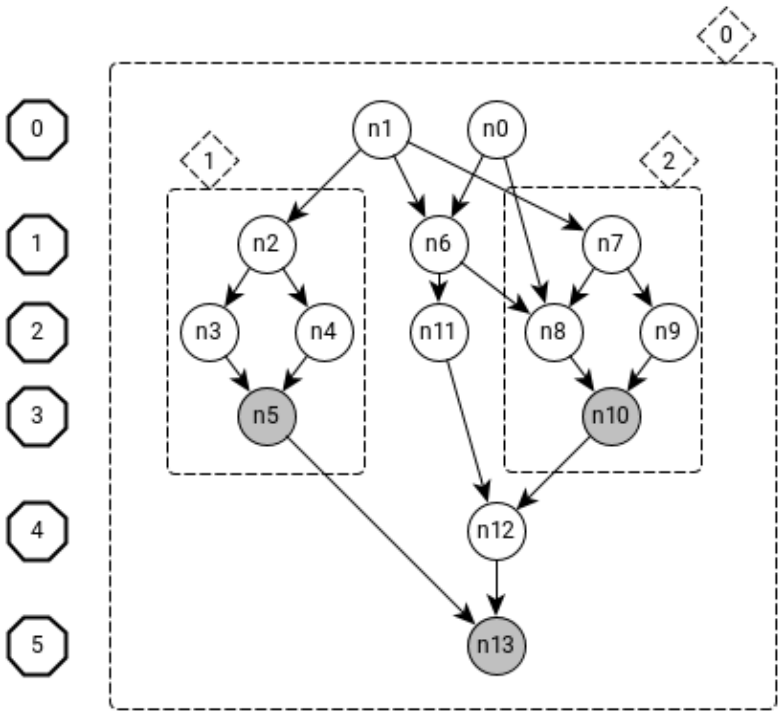



**Fig. 2.** Information graph with highlighted delayed lists and tiers

**Рис. 2.** Информационный граф с выделенными задержанными списками и ярусами

Алгоритм трансляции состоит из ряда этапов:

1) Разметка узлов типами на основе типа аргумента функции и правил модели вычислений.
2) Разметка узлов именами переменных, соответствующих узлам ГПЗ в генерируемом коде на императивном языке программирования.
3) Начальная разметка узлов флагами трансляции. Используются следующие флаги:
   - ***NOT_TRANSLATED*** – необработанный узел;
   - ***NO_TRANSLATABLE*** – узел, не требующий преобразования;
   - ***TRANSLATED*** – обработанный узел.
   - ***SWITCH_RESULT*** – метка для узлов, являющихся результатами вычисления ветвей алгоритма при трансляции условных операторов.
4) Подготовка узлов к трансляции. При этом в ГПЗ выделяются и размечаются подграфы, задающие определенные синтаксические конструкции языка C++.
5) Трансляция констант.
6) Трансляция выделенных подграфов для синтаксических конструкций по описанным выше шаблонам.

## Заключение

В работе описаны проблемы и особенности преобразования функционально-потоковых программ на основе реверсивных информационных графов, задающих максимальный параллелизм, в императивную форму. Приведены некоторые детали реализации такой системы, преобразующей программы языка Пифагор в язык C++. Предложены:

1) формат описания типов функций функционально-потокового языка программирования;
2) алгоритм преобразования, основанный на поиске в реверсивных информационных графах конструкций, соответствующих заданному множеству шаблонов;
3) структуры данных, обеспечивающие эффективную реализацию такого алгоритма.

Разработанная система позволяет преобразовывать достаточно широкий класс программ и является основой для дальнейшего анализа конструкций ФПП-программ, которые имеет смысл выполнять параллельно на конкретной целевой архитектуре.

## References


[1] K. Vivek, *Parallel Computing Architectures and APIs: IoT Big Data Stream Processing*. Dec. 2019, ISBN: 9781351029223. DOI: 10.1201/9781351029223.

[2] I. Levin, A. I. Dordopulo, and V. A. Gudkov, "Programming of reconfigurable computing nodes in the COLAMO language", *Training manual. Taganrog: Publishing house of TTI SFEDU. In Russian*, 2011.

[3] A. I. Dordopulo and I. I. Levin, "Resource-independent programming of hybrid reconfigurable computing systems", in *Russian Supercomputing Days. In Russian*, 2017, pp. 714–723.

[4] V. Kasyanov, "Sisal 3.2: functional language for scientific parallel programming", *Enterprise Information Systems*, vol. 7, no. 2, pp. 227–236, 2013.

[5] A. I. Legalov, "Functional language for creating architecturally independent parallel programs", *Computing technologies*, vol. 10, no. 1, 2005.

[6] I. I. Levin, A. I. Dordopulo, I. V. Pisarenko, and A. K. Melnikov, "An approach to architecture-independent programming of computing systems based on the aspect-oriented Set@ language", *Proceedings of the Southern Federal University. Technical sciences. In Russian*, vol. 197, no. 3, 2018.

[7] A. I. Legalov, V. S. Vasilev, I. V. Matkovskii, and M. S. Ushakova, "A toolkit for the development of data-driven functional parallel programmes", in *International Conference on Parallel Computational Technologies*, Springer, 2018, pp. 16–30.

[8] V. S. Vasilev and A. I. Legalov, "Loop-invariant Optimization in the Pifagor Language", *Automatic Control and Computer Sciences*, vol. 52, no. 7, pp. 843–849, 2018.

[9] U. V. Udalova, A. I. Legalov, and N. U. Sirotinina, "Methods for debugging and verifying functional-stream parallel programs", *Journal of the Siberian Federal University. Equipment and technologies*, vol. 4, no. 2, 2011.

[10] M. S. Ushakova and A. I. Legalov, "Verification of Programs with Mutual Recursion in Pifagor Language", *Automatic Control and Computer Sciences*, vol. 52, no. 7, pp. 850–866, 2018.

[11] S. V. Zykov, *Fundamentals of Modern Programming. Development of heterogeneous systems in an Internet-oriented environment*. IPR Media, 2017. DOI: 10.23682/62072.

[12] A. I. Legalov, I. A. Legalov, and I. V. Matkovsky, "Specifics of semantics of a statically typed language of functional and dataflow parallel programming", in *Scientific Conference Scientific Service on the Internet*, 2019, pp. 489–500.

[13] A. I. Legalov, I. V. Matkovsky, M. S. Ushakova, and D. S. Romanova, "Dynamically Changing Parallelism with the Asynchronous Sequential Data Flows", *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 164–179, 2020.

[14] O. V. Nepomnyashchiy, I. N. Ryzhenko, and A. I. Legalov, "Method of architecture-independent high-level synthesis of VLSI", *Proceedings of the Southern Federal University. Technical sciences*, vol. 202, no. 8, 2018.

[15] O. V. Nepomnyashchiy, I. N. Ryzhenko, and A. I. Legalov, "Methods, algorithms, and software tools for architecturally independent high-level synthesis of single-chip digital systems", in *Supercomputing Technologies (SCT-2018)*, 2018, pp. 104–109.

[16] M. S. Ushakova, "Data type semantics of the dataflow parallel programming language Pifagor", *Educational resources and technologies*, vol. 14, no. 2, 2016.

[17] I. V. Matkovsky and A. I. Legalov, "Instrumental support for the translation and execution of functional-stream parallel programs", *Polzunovsky vestnik*, no. 2, pp. 49–52, 2013.

[18] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization", *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 9, no. 3, pp. 319–349, 1987.

[19] V. M. Bakanov, "Software tools for analyzing the information structure of algorithms based on their information graphs", in *Parallel Computing Technologies (PaVT ' 2016)*, 2016, pp. 432–441.