ISSN 1818-1015
eISSN 2313-5417

2020 Volume 27 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

2020 Том 27 № 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». Зав. редакцией – М. С. Каряева, корректор английских текстов – М. С. Комар. Подписано в печать 22.06.2020. Дата выхода в свет 30.06.2020. Формат 200×265 мм. Объем 122 с. Тираж 46 экз. Свободная цена. Заказ 020/020. 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. 27, NO. 2, 2020
journal homepage: www.mais-journal.ru

COMPUTER SYSTEM ORGANIZATION

On the Detection of Exploitation of Vulnerabilities Leading to the Execution of a Malicious Code

Y. V. Kosolapov[1]

DOI: 10.18255/1818-1015-2020-2-138-151

[1]Southern Federal University, 8a Milchakova str., Rostov-on-Don 344090, Russia.

MSC2020: 68M25
Research article
Full text in Russian

Received March 9, 2019
After revision March 23, 2020
Accepted March 25, 2020

Software protection from exploitation of possible unknown vulnerabilities can be performed both by searching (for example, using symbolic execution) and subsequent elimination of the vulnerabilities and by using detection and / or intrusion prevention systems. In the latter case, this problem is usually solved by forming a profile of a normal behavior and deviation from normal behavior over a predetermined threshold is regarded as an anomaly or an attack. In this paper, the task is to protect a given software P from exploiting unknown vulnerabilities. For this aim a method is proposed for constructing a profile of the normal execution of the program P, in which, in addition to a set of legal chains of system and library functions, it is proposed to take into account the distances between adjacent function calls. At the same time, a profile is formed for each program. It is assumed that taking into account the distances between function calls will reveal shell code execution using system and / or library function calls. An algorithm and a system for detecting abnormal code execution are proposed. The work carried out experiments in the case when P is the FireFox browser. During the experiments the possibility of applying the developed algorithm to identify abnormal behavior when launching publicly available exploits was investigated.

Keywords: system calls; library calls; software vulnerability.

INFORMATION ABOUT THE AUTHORS

Yury V. Kosolapov
correspondence author
orcid.org/0000-0002-1491-524X. E-mail: itaim@mail.ru
PhD.

For citation: Y. V. Kosolapov, “On the Detection of Exploitation of Vulnerabilities Leading to the Execution of a Malicious Code”, *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 138-151, 2020.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 27, № 2, 2020
сайт журнала: www.mais-journal.ru

COMPUTER SYSTEM ORGANIZATION

Об обнаружении эксплуатации уязвимостей, приводящей к запуску вредоносного кода

Ю. В. Косолапов[1]

DOI: 10.18255/1818-1015-2020-2-138-151

[1]Южный Федеральный Университет, ул. Мильчакова, 8а, г. Ростов-на-Дону, 344090 Россия.

УДК 517.9
Научная статья
Полный текст на русском языке

Получена 9 марта 2019 г.
После доработки 23 марта 2020 г.
Принята к публикации 25 марта 2020 г.

Задача защиты программного обеспечения от эксплуатации возможных неизвестных уязвимостей может решаться как путем поиска (например, с помощью символьного исполнения) и последующего устранения уязвимостей, так и путем использования систем обнаружения и/или предотвращения вторжений. В последнем случае эта задача решается обычно путем формирования профиля нормального выполнения программ, а недопустимое отклонение от нормального состояния расценивается как аномалия или атака. В настоящей работе рассматривается задача защиты заданного исполнимого файла (программы) P от эксплуатации неизвестных уязвимостей в нем. Для этого предлагается способ построения профиля нормального выполнения программы P, в котором кроме набора легальных цепочек системных и библиотечных функций длины l учитывается расстояние между соседними вызовами функций, вычисляемое как разность адресов вызова соответствующих функций. Учет расстояний между вызовами функций позволяет выявлять исполнение вредоносного шеллкода, использующего вызовы системных и/или библиотечных функций, если хотя бы один из используемых в шеллкоде вызовов находится на нетипичном для программы P расстоянии от предыдущего вызова. В работе строится алгоритм и система обнаружения аномального выполнения кода и проводятся эксперименты в случае, когда P — браузер FireFox для операционной системы Windows.

Ключевые слова: системные вызовы; вызовы библиотек; уязвимости программного обеспечения.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Юрий Владимирович Косолапов
автор для корреспонденции
orcid.org/0000-0002-1491-524X. E-mail: itaim@mail.ru
канд. техн. наук.

Для цитирования: Y. V. Kosolapov, “On the Detection of Exploitation of Vulnerabilities Leading to the Execution of a Malicious Code”, *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 138-151, 2020.

Введение и постановка задачи

Выявление вредоносного программного обеспечения (ВПО) на основе аномального выполнения обладает преимуществом перед выявлением ВПО на основе его сигнатуры: первые позволяют обнаружить новые, еще не изученные образцы ВПО. В системах на основе выявления аномального поведения выделяются наборы признаков, которые отличают поведение легитимной программы от поведения ВПО [1]. Признаки в таких наборах часто строятся на основе последовательности API-вызовов (API — application program interface) или графа вызовов функций, используемых в ПО [2]. Например, в [3] мониторинг API-вызовов операционной системы Windows (функций из библиотек `kernel32.dll`, `advapi32.dll`, `gdi32.dll`, `comct32.dll`, `user32.dll`, `shell32.dll`, `ntdll.dll`) используется для классификации *исполнимых файлов* ВПО на пять различных классов: Worm (червь), Trojan-Downloader (загрузчик), Trojan-Spy (шпионское ПО), Trojan-Dropper (инсталлятор) и Backdoor (бэкдор). Для каждого из этих классов на основе взаимодействия ВПО с сетью, реестром, системными сервисами и т.п., происходящего посредством вызовов соответствующих API-функций, в [3] находится признак класса. В основе построения признака, сравнения и классификации лежит техника нечеткого хеширования (fuzzy hashing). В отличие от [3], где решение о классе ВПО принимается во время исполнения ВПО (т.е. динамически), в [4] решение о наличии вредоносного исполнения принимается на основе статического анализа графа API-вызовов. Для получения информации о вызываемых системных функциях используется дизассемблер IDA Pro. Заметим, что статический анализ затруднен для обфусцированных файлов. В [4] на основе техники DCFS (Document Class Frequency Selection) выбирают комбинации (n-граммы) API-функций, которые характерны для ВПО, а также комбинации API-функций, которые характерны для легитимных программ. На основе двух этих наборов строится вектор признаков (характеристик) для обучения SVM-классификатора. Статический анализ последовательности API-вызовов применен также и в [5], где классификатор строится на основе вычисления показателя похожести графов вызовов. В работе [6] для выявления ВПО и определения класса ВПО используются методы процессной аналитики (process mining): выполнение исполнимого файла описывается с помощью декларативной модели — конечного автомата, в котором переходы между состояниями помечены системными вызовами. Алгоритм выявления основан на вычислении расстояния между моделью, построенной для исследуемого исполнимого файла, и моделями известных семейств ВПО, и далее применении одного из алгоритмов классификации. Подробную классификацию систем обнаружения аномалий на основе системных вызовов можно найти, например, в [7]. Кроме признаков, строящихся исключительно на коде ВПО или на основе генерируемой последовательности машинных команд/API-вызовов, могут использоваться признаки, основанные на том, как ВПО преобразует данные. Например, для вирусов-шифровальщиков признаком выполнения может являться энтропия записываемых ими данных, которая для зашифрованных данных обычно выше энтропии незашифрованных данных.

Выявление аномального поведения может использоваться не только для обнаружения ВПО, запущенного в системе в виде отдельного исполнимого файла, но и для обнаружения эксплоитов, нарушающих работу легитимного ПО. Однако выявление исполнения кода, эксплуатирующего уязвимость, намного сложнее, чем выявление вредоносного кода, на который обычно передается управление эксплоитом [8]. Для выявления эксплоитов также необходимо иметь совокупность признаков, и «нормальные» значения этих признаков, которые бы характеризовали выполнение программы в соответствии с заложенными в нее алгоритмами. Отклонение значений от «нормальных» может свидетельствовать как о запуске эксплоита (истинное обнаружение), так и о том, что программа получила на вход легитимные данные, которые не были использованы на этапе обучения (ложное обнаружение). Шеллкод, на который передается управление эксплоитом после успешной эксплуатации уязвимости (например, при переполнении буфера в стеке или в куче), часто вызывает API-функцию, например, функцию выделения памяти `VirtualAlloc` в операци-

онной системе Windows. В общем случае, вызываемая функция может быть любой, в том числе и библиотечной, так как различные техники повторного использования исполнимого кода позволяют вызвать практически любую такую функцию (описание техник повторного использования исполнимого кода можно найти, например, в работах [9—11]). Так как выполнение эксплоита может привести к нарушению обычного порядка вызова системных и библиотечных функций, то признаком аномального выполнения легитимного ПО может являться нетипичная последовательность системных и библиотечных функций. Этот подход обнаружения аномалий подробно описан в [2], однако он может быть нивелирован в ряде случаев мимикрирующими атаками [12].

Так как после завершения загрузки статических библиотек адреса функций не меняются, то признаком аномального поведения легитимного ПО может быть расстояние между соседними вызовами функций в трассе вызовов, полученной после запуска программы (под расстоянием понимается разность адресов вызова функций). Сформированный для существующей в легитимном ПО уязвимости шеллкод, который вызывает системную или библиотечную функцию, может находиться на нетипичном для нормального выполнения расстоянии от вызова предыдущей или до вызова следующей отслеживаемой функции. Это относится и к шеллкоду на основе повторного использования исполнимого кода, и к шеллкоду, записываемому в динамически выделенную *исполнимую* память. В настоящей работе ставится задача оценки возможности применения расстояния между API-вызовами в качестве признака для обнаружения факта запуска эксплоита в легитимной программе. В первом разделе описывается способ построения профиля программы и алгоритм обнаружения, а во втором – описаны результаты экспериментов, проведенных для браузера FireFox.

1. Модель обнаружения аномального поведения

В настоящей работе, с одной стороны, предлагается по трассе системных и библиотечных вызовов строить цепочки вызовов, как описано в [2], а с другой стороны – с целью противодействия атакам маскировки под легальные цепочки вызовов, описанным в [12], учитывать расстояние между соседними системными вызовами и/или библиотечными вызовами. При внедрении нелегитимного кода (в том числе, с помощью техник повторного использования исполнимого кода [9—11]) появляется точка вызова функции (системной или библиотечной), расстояние от которой (или до которой) может быть *нетипичным*. Наличие нетипичного расстояния может служить признаком заражения и использоваться для защиты конкретных приложений (профиль типичных расстояний индивидуален для каждой программы). Защищаемыми приложениями могут быть, например, просмоторщики pdf-файлов, редакторы документов, браузеры.

1.1. Алгоритмы выявления аномального поведения

Пусть $\mathcal{L}(P)$ – множество имен библиотечных функций и функций ядра, используемых программой P. Заметим, что при запуске исполнимого файла (программы P) в виртуальную память процесса загружаются используемые им библиотеки. В случае, когда задействована технология рандомизации адресного пространства (Address Space Layout Randomization — ASLR), в разные моменты запуска исполнимого файла одна и та же библиотека может быть загружена по разным адресам. Отметим, что технология ASLR направлена на защиту от атак типа повторного использования исполнимого кода, однако такая защита может быть нивелирована, если в исполняемой программе удается найти и использовать уязвимость типа раскрытия памяти, при эксплуатации которой атакующий получает информацию об адресах загрузки библиотек [13]. Введем для удобства вспомогательную функцию B(`addr`), которая по адресу `addr` возвращает адрес модуля, адресному пространству которого этот адрес принадлежит. Пусть $\mathrm{Path}_t(P(I))$ — последовательность системных вызовов и вызовов библиотек *одного потока* программы P, загруженной в память в момент времени t и запускаемой с входными данными I:

$$\mathrm{Path}_t(P(I)) = (f_1^{t,I}, ..., f_{N_{t,I}}^{t,I}), \tag{1}$$

где API-вызов $f_k^{t,I}$ с порядковым номером k представим в виде тройки:

$$f_k^{t,I} = (B(\mathtt{ca}_k^{t,I}), \mathtt{ca}_k^{t,I}, n_k^{t,I}), \tag{2}$$

$B(\mathtt{ca}_k^{t,I})$ — адрес, по которому загружен исполнимый модуль (библиотека) в виртуальную память, содержащий по адресу $\mathtt{ca}_k^{t,I}$ вызов функции с именем $n_k^{t,I} (\in \mathcal{L}(P))$. Для $t_1 \neq t_2$ последовательности API-вызовов $\mathrm{Path}_{t_1}(P(I))$ и $\mathrm{Path}_{t_2}(P(I))$ (при одних и тех же входных параметрах программы) в общем случае, в силу ASLR, отличаются адресами $B(\mathtt{ca})$ и $\mathtt{ca}$ (см. таблицу 1).

Table 1. An example of a single program execution trace at times t_1 and t_2

Таблица 1. Пример трассы выполнения одной программы в моменты времени t_1 и t_2

N	$\mathrm{Path}_{t_1}(P(I))$	$\mathrm{Path}_{t_2}(P(I))$
1	$(B(\mathtt{ca}_1^{t_1,I}), \mathtt{ca}_1^{t_1,I}, n_1^{t_1,I})$	$(B(\mathtt{ca}_1^{t_2,I}), \mathtt{ca}_1^{t_2,I}, n_1^{t_2,I})$
2	$(B(\mathtt{ca}_2^{t_1,I}), \mathtt{ca}_2^{t_1,I}, n_2^{t_1,I})$	$(B(\mathtt{ca}_2^{t_2,I}), \mathtt{ca}_2^{t_2,I}, n_1^{t_2,I})$
3	$(B(\mathtt{ca}_3^{t_1,I}), \mathtt{ca}_3^{t_1,I}, n_3^{t_1,I})$	$(B(\mathtt{ca}_3^{t_2,I}), \mathtt{ca}_3^{t_2,I}, n_1^{t_2,I})$
⋮	⋮	⋮

Вообще говоря, $n_k^{t_1,I} \neq n_k^{t_2,I}$ в общем случае, при одних и тех же входных данных I (когда, исполнение программы зависит не только от входных данных, но и, например, от системного счетчика времени или значения на выходе генератора псевдослучайных чисел). Однако без потери общности, предполагается, что в таблице 1 вызовы с порядковым номером k (столбец N), как в момент времени t_1, так и в момент времени t_2 выполняются из одного и того же модуля (другими словами, по адресам $B(\mathtt{ca}_k^{t_1,I})$ и $B(\mathtt{ca}_k^{t_2,I})$ загружен один и тот же модуль, причем при разных k модули могут быть разными). Отсюда получаем, что в общем случае $d_{k,k+1}^{t_1,I} = \mathtt{ca}_{k+1}^{t_1,I} - \mathtt{ca}_k^{t_1,I} \neq \mathtt{ca}_{k+1}^{t_2,I} - \mathtt{ca}_k^{t_2,I} = d_{k,k+1}^{t_2,I}$ для $n_k^{t_1,I} = n_k^{t_2,I}$ и $n_{k+1}^{t_1,I} = n_{k+1}^{t_2,I}$. С другой стороны, для любого k при $n_k^{t_1,I} = n_k^{t_2,I}$ выполняется равенство $\mathtt{ca}_k^{t_1,I} - B(\mathtt{ca}_k^{t_1,I}) = \mathtt{ca}_k^{t_2,I} - B(\mathtt{ca}_k^{t_2,I})$. Тогда

$$\mathtt{ca}_k^{t_2,I} = \mathtt{ca}_k^{t_1,I} + (B(\mathtt{ca}_k^{t_2,I}) - B(\mathtt{ca}_k^{t_1,I})) = \mathtt{ca}_k^{t_1,I} + \Delta_k^{t_1,t_2,I} = M_{t_2}(\mathtt{ca}_k^{t_1,I}). \tag{3}$$

Поэтому,

$$d_{k,k+1}^{t_2,I} = M_{t_2}(\mathtt{ca}_{k+1}^{t_1,I}) - M_{t_2}(\mathtt{ca}_k^{t_1,I}) = d_{k,k+1}^{t_1,I} + \Delta_{k+1}^{t_1,t_2,I} - \Delta_k^{t_1,t_2,I}. \tag{4}$$

Таким образом, если накопление информации (обучение) о расстояниях между вызовами проводится в момент времени t_1, а проверка – в момент времени t_2, то выражение (3) позволяет спроецировать наблюдаемые в момент тестирования значения адресов на те адреса загрузки библиотек, которые были в момент обучения. Для $(n, m) \in \mathcal{L}(P) \times \mathcal{L}(P)$ будем писать $(n, m) \propto_d \mathrm{Path}_t(P(I))$, если найдется такое k, что $n = n_k^{t,I}$, $m = n_{k+1}^{t,I}$ и $d = \mathtt{ca}_{k+1}^{t,I} - \mathtt{ca}_k^{t,I}$ для троек $(B(\mathtt{ca}_k^{t,I}), \mathtt{ca}_k^{t,I}, n_k^{t,I})$ и $(B(\mathtt{ca}_{k+1}^{t,I}), \mathtt{ca}_{k+1}^{t,I}, n_{k+1}^{t,I})$. Пусть $\mathcal{I}(P)$ — множество значений *легитимных* входных данных программы P (множество тех данных, на которых строится профиль нормального поведения программы). Для каждой пары $(n, m) \in \mathcal{L}(P) \times \mathcal{L}(P)$ рассмотрим множество

$$D_{(n,m)}^t = \{d \in \mathbb{Z} | \exists I \in \mathcal{I}(P) \,:\, (n, m) \propto_d \mathrm{Path}_t(P(I))\}.$$

Другими словами, $D_{(n,m)}^t$ — множество возможных расстояний между API-функциями n и m, вызываемыми друг за другом (функция m вызывается после функции n). Отметим, что в общем случае $D_{(n,m)}^t \neq D_{(m,n)}^t$. *Профилем расстояний* программы P назовем множество

$$\mathcal{D}^t(P) = \left\{ D_{(n,m)}^t \,:\, (n, m) \in \mathcal{L}(P) \times \mathcal{L}(P) \right\}, \tag{5}$$

построенное *по всем потокам* программы. Пусть $l^t_{min}(P)$ и $l^t_{max}(P)$ — минимальная и максимальная длина списка расстояний в профиле $\mathcal{D}^t(P)$. Максимальное и минимальное (по профилю (5)) значение расстояния между соседними вызовами для программы P в момент запуска t_1 обозначим соответственно $d^{t_1}_{max}(P)$ и $d^{t_1}_{min}(P)$. Очевидно, что если спроецированное наблюдаемое значение расстояния между соседними API-вызовами не принадлежит диапазону

$$[d^{t_1}_{min}(P) \,:\, d^{t_1}_{max}(P)] = \{d^{t_1}_{min}(P), d^{t_1}_{min}(P)+1, ..., d^{t_1}_{max}(P)\},$$

то это свидетельствует об аномальном выполнении. *Профилем цепочек* порядка l программы P, в соответствии с [2], назовем множество

$$C_t(P,l) = \left\{(n_1,...,n_l) | \exists I \in \mathcal{I}(P) \exists k \in \mathbb{N} \forall r \in \{1,...,l\} \,:\, n^{t,I}_{k+r} = n_r\right\}. \tag{6}$$

Алгоритм *CheckTrace* (см. алгоритм 1) выявляет нетипичное выполнение программы P на основе профиля расстояний (5) и профиля цепочек (6).

Алгоритм 1. *CheckTrace*

Исходные параметры: 1) последовательность $\mathrm{Path}_{t_2}(P(I))$ вида (1) длины n_I ($I \notin \mathcal{I}(P)$),
2) профиль $\mathcal{D}_{t_1}(P)$ вида (5) и профиль $\mathcal{C}_{t_1}(P,l)$ вида (6)

Результат : Сообщение о нетипичной (`not typical`) или типичной (`typical`) последовательности API-вызовов

```
result = typical
цикл k = 1, ..., n_I выполнять
    если k ⩾ l и (n^{t2,I}_{k-l+1}, ..., n^{t2,I}_k) ∉ C_{t1}(P, l) тогда
        result = not typical
        Выйти из цикла
    если k ⩽ n_I − 1 тогда
        d = d^{t2,I}_{k,k+1} − Δ^{t1,t2,I}_{k+1} + Δ^{t1,t2,I}_k
        если d ∈ [d^{t1}_min(P) : d^{t1}_max(P)] тогда
            если d ∉ D_{(f^{t2,I}_k, f^{t2,I}_{k+1})} тогда
                result = not typical
        иначе
            result = not typical
            Выйти из цикла
возвратить result
```

В алгоритме сначала выполняется проверка, является ли текущая цепочка вызовов длины l из трассы вызовов $\mathrm{Path}_{t_2}(P(I))$ типичной, то есть содержится ли эта цепочка в профиле $\mathcal{C}_{t_1}(P,l)$. Отсутствие цепочки свидетельствует об отклонении от нормального исполнения. В случае, если цепочка присутствует в профиле, то проверяется является ли типичным расстояние между соседними API-вызовами. Отметим, в случае $l = 2$, проверка на типичность цепочки является избыточной, так как в этом случае $D_{n,m} = \varnothing$ для нетипичной пары (n, m), и достаточно проверки по профилю расстояний.

Утверждение 1. *Пусть* $[A_1 \,:\, A_2]$ *— диапазон адресов виртуальной памяти в момент времени* t_1*, используемый для загрузки кода программы* P *и требуемых модулей/библиотек,* $[E_1 \,:\, E_2]$ *— диапазон*

адресов памяти в момент времени t_2 отслеживаемого API-вызова из шеллкода, содержащегося во входных данных I, $[A_1 : A_2] \cap [M_{t_2}^{-1}(E_1) : M_{t_2}^{-1}(E_2)] = \varnothing$. Если $A_2 - A_1 < M_{t_2}^{-1}(E_1) - A_2$ либо $A_2 - A_1 < A_1 - M_{t_2}^{-1}(E_2)$, то

$$CheckTrace(\mathrm{Path}_{t_2}(P(I)), \boldsymbol{D}_{t_1}(P), C_{t_1}(P, l), 1) = \texttt{not typical}$$

для любого $t_2 \geqslant t_1$.

Доказательство. Доказательство утверждения вытекает из того, что расстояние между легитимными вызовами будет не более $A_2 - A_1$, а шеллкод обращается к API-функции с адреса, который находится от предшествующего API-вызова на расстоянии d, большем $A_2 - A_1$. Следовательно, число d не будет в списке возможных расстояний и алгоритм $CheckTrace$ вернет результат проверки `not typical`. Этот результат будет возвращен в любой момент времени $t_2 \geqslant t_1$, так как выражение (4) позволяет перейти от адресации в момент времени t_2 к адресации в момент t_1. □

Отметим, что в операционной системе Windows диапазон адресов загрузки модулей библиотек и диапазон области кучи, в которой часто размещается шеллкод, пересекаются, поэтому условия утверждения 1 для этой операционной системы обычно не выполняются. Следующие два утверждения доказываются по аналогии с утверждением 1.

Утверждение 2. *Пусть $[A_1 : A_2]$ — диапазон адресов виртуальной памяти в момент времени t_1, из области которого отслеживаются API-вызовы программы P, $[E_1 : E_2]$ — диапазон адресов памяти в момент времени t_2 отслеживаемого API-вызова из шеллкода, содержащегося во входных данных I, $[A_1 : A_2] \cap [M_{t_2}^{-1}(E_1) : M_{t_2}^{-1}(E_2)] = \varnothing$. Если $A_2 - A_1 < M_{t_2}^{-1}(E_1) - A_2$ либо $A_2 - A_1 < A_1 - M_{t_2}^{-1}(E_2)$, то*

$$CheckTrace(\mathrm{Path}_{t_2}(P(I)), \boldsymbol{D}_{t_1}(P), C_{t_1}(P, l), 1) = \texttt{not typical}$$

для любого $t_2 \geqslant t_1$.

Утверждение 3. *Пусть $[A_1 : A_2]$ — диапазон адресов памяти в момент времени t_1, используемый для загрузки кода программы P и требуемых модулей/библиотек, D — максимальное расстояние между отслеживаемыми API-вызовами программы P в момент времени t_1 для всех легитимных входных данных, $[E_1 : E_2]$ — диапазон адресов памяти в момент времени t_2 отслеживаемого API-вызова из шеллкода, содержащегося во входных данных I, $[A_1 : A_2] \cap [M_{t_2}^{-1}(E_1) : M_{t_2}^{-1}(E_2)] = \varnothing$. Если $D < M_{t_2}^{-1}(E_1) - A_2$ либо $D < A_1 - M_{t_2}^{-1}(E_2)$, то*

$$CheckTrace(\mathrm{Path}_{t_2}(P(I)), \boldsymbol{D}_{t_1}(P), C_{t_1}(P, l), 1) = \texttt{not typical}$$

для любого $t_2 \geqslant t_1$.

Пусть $\boldsymbol{I}(P)$ и $\boldsymbol{I}^E(P)$ — множество легитимных и нелегитимных входных значений программы P, $\boldsymbol{I}^E(P) \cap \boldsymbol{I}(P) = \varnothing$, $N^L = |\boldsymbol{I}(P)|$, $N^E = |\boldsymbol{I}^E(P)|$, A – алгоритм обнаружения, использующий профиль $\boldsymbol{D}_{t_1}(P)$ вида (5) и профиль $C_{t_1}(P, l)$ вида (6),

$$J^A(I) = \begin{cases} 1, & A(\mathrm{Path}_{t_2}(P(I))) = \texttt{not typical}, \\ 0, & A(\mathrm{Path}_{t_2}(P(I))) = \texttt{typical}. \end{cases}$$

Важными характеристиками систем обнаружения аномалий являются: вероятность ложного обнаружения P_{FP}^A, вероятность истинного обнаружения P_{TP}^A (чувствительность обнаружения или уровень обнаружения) [1]:

$$P_{FP}^A = \frac{\sum_{I \in \boldsymbol{I}(P)} J^A(I)}{N^L}, P_{TP}^A = \frac{\sum_{I \in \boldsymbol{I}^E(P)} J^A(I)}{N^E}.$$

Для возможного снижения вероятности ложного обнаружения разработан также алгоритм обнаружения $CheckTraceBack$ (см. алгоритм 2), в котором учитывается расстояние не до одного предыдущего вызова, а расстояния до $T+1$ предыдущих вызовов. Именно, в случае, когда обнаруживается нетипичное расстояние между соседними вызовами (в случае, когда алгоритм $CheckTrace$ возвращает результат `not typical`), предлагается вычислять двоичный вектор вида

$$\mathbf{b}_{f_k^{t_2,I},f_{k+1}^{t_2,I}} = (b_1, ..., b_T) \in \{0,1\}^T, \tag{7}$$

i-ая координата ($i \in \{1, ..., T\}$) которого определяется по правилу:

$$b_i = \begin{cases} 1, d_{k-i,k+1}^{t_2,I} - \Delta_{k+1}^{t_1,t_2,I} + \Delta_{k-i}^{t_1,t_2,I} \notin D_{f_{k-i}^{t_2,I},f_{k+1}^{t_2,I}} \\ 0, \text{ иначе} \end{cases}.$$

В алгоритме $CheckTraceBack$ аномальным поведением считается вызов $f_{k+1}^{t_2,I}$, для которого расстояние до предыдущего вызова является нетипичным и вес Хэмминга вектора (7) равен T: все расстояния до вызова $f_{k+1}^{t_2,I}$ от предыдущих $T+1$ вызовов являются нетипичными (что представляется характерным при выполнении эксплоита). Отметим, что в общем случае можно ввести порог на вес вектора (7), при превышении которого вызов считается нетипичным. Однако это обобщение в настоящей работе не рассматривается.

Отметим ряд ограничений разработанных алгоритмов. Во-первых, отслеживание API-вызовов для определенного диапазона адресов (как в утверждении 2) может привести к пропуску запуска эксплоита, даже если API-функция является отслеживаемой (например, если вызов API-функции выполнен из библиотеки, вызовы из которой не отслеживаются). Во-вторых, если эксплоит использует функции, не входящие в $\mathcal{L}(P)$, либо вообще не использует API-вызовы, то его выполнение не будет обнаружено (целью таких эксплоитов обычно является остановка работы приложения). В-третьих, предлагаемый способ защиты не обнаруживает запуск вредоносного кода, если эксплуатация уязвимости не предполагает внедрение/запись исполнимого кода в память уязвимой программы. В частности, атака на основе динамического обмена данными (Dynamic Data Exchange — DDE), описанная в [14], не будет выявлена, так как в этом случае нормальное исполнение программы неотличимо от аномального только на основе вызываемых API-функций и расстояний между ними. Для выявления таких атак необходим анализ входных параметров вызываемых функций, что усложняет алгоритм обнаружения.

Создатель вредоносного кода может попытаться замаскировать выполнение своего кода под легальный профиль: для маскировки необходимо, чтобы адрес инструкции, вызывающей API-функцию, был на легитимном расстоянии от предыдущей вызванной API-функции и на легитимном расстоянии до следующей API-функции, а также чтобы результирующая последовательность API-вызовов находилась в базе легитимных цепочек. Если информация об адресах загрузки модулей в момент времени t_1 (в момент обучения) доступна создателю вредоносного кода, то он может, используя подходящую уязвимость, в результате эксплуатации которой в момент времени t_2 (в момент эксплуатации) происходит утечка информации о базовых адресах загруженных модулей, подготовить эксплоит, в котором вызов API-функции будет находиться на легитимном расстоянии (вычисленном в соответствии с (4)) от предыдущего API-вызова и на легитимном расстоянии до следующего API-вызова. Подбор подходящего расстояния может быть выполнен с помощью вставки команд `nop` в машинный код. Поэтому информация об адресах загрузки модулей в момент t_1 должна быть доступна только системе защиты. При удаленной эксплуатации эта информация, как правило, неизвестна создателю кода. При локальной эксплуатации эту информацию можно сделать труднодоступной, например, храня ее в зашифрованном файле и расшифровывая только в момент загрузки системы защиты (ключ расшифрования, например, может храниться в исполнимом файле

Алгоритм 2. *CheckTraceBack*

Исходные параметры: 1) последовательность $\mathrm{Path}_{t_2}(P(I))$ вида (1) длины n_I ($I \notin \mathcal{I}(P)$), 2) профиль $\mathcal{D}_{t_1}(P)$ вида (5) и профиль $\mathcal{C}_{t_1}(P, l)$ вида (6), порог обнаружения T

Результат : Сообщение о нетипичной (`not typical`) или типичной (`typical`) последовательности API-вызовов

```
result = typical
цикл k = 1, ..., n_I выполнять
    если k ⩾ l и (n^{t2,I}_{k−l+1}, ..., n^{t2,I}_k) ∉ C_{t1}(P, l) тогда
        result = not typical
        Выйти из цикла
    если k ⩽ n_I − 1 тогда
        d = d^{t2,I}_{k,k+1} − Δ^{t1,t2,I}_{k+1} + Δ^{t1,t2,I}_k
        если d ∈ [d^{t1}_min(P) : d^{t1}_max(P)] тогда
            если d ∉ D_{f^{t2,I}_k, f^{t2,I}_{k+1}} тогда
                result = not typical
        иначе
            result = not typical
        если result = not typical тогда
            Вычислить вектор b вида (7)
            если wt(b) = T тогда
                Выйти из цикла
            иначе
                result = typical
                Добавить расстояние d в профиль D_{t1}(P) для пары (f^{t2,I}_k, f^{t2,I}_{k+1}).
возвратить result
```

системы защиты, защищенном подходящими методами обфускации [15]). Отметим, что даже в случае, когда информация об адресах загрузки модулей известна разработчику эксплоита, написание кода таким образом, чтобы вызываемые API-функции, с одной стороны, не приводили к появлению нетипичных цепочек (то есть чтобы цепочки принадлежали $\mathcal{C}_{t_1}(P, l)$), а с другой стороны, находились на легитимных расстояниях в типичных цепочках (то есть чтобы расстояния между вызовами принадлежали $\mathcal{D}_{t_1}(P)$), является сложной задачей. Также представляется, что список отслеживаемых API-функций должен оставаться в секрете: разработчику эксплоита для подбора расстояния необходима информация о том, какой API-вызов будет перед API-вызовом в шеллкоде.

1.2. Схема системы обнаружения аномального поведения

Схема системы обнаружения аномального поведения показана на рис. 1. Как и в большинстве систем выявления аномального поведения [1], в предлагаемой системе предусмотрен этап обучения, в ходе которого строятся профили $\mathcal{D}_t(P)$ и $\mathcal{C}_t(P, l)$ защищаемой программы. Входными данными для этого этапа являются набор $\mathcal{I}(P)$ (передается в блок сбора данных), набор $\mathcal{L}(P)$ (передается в блок настройки параметров) и натуральное число l (передается в блок построения профилей).

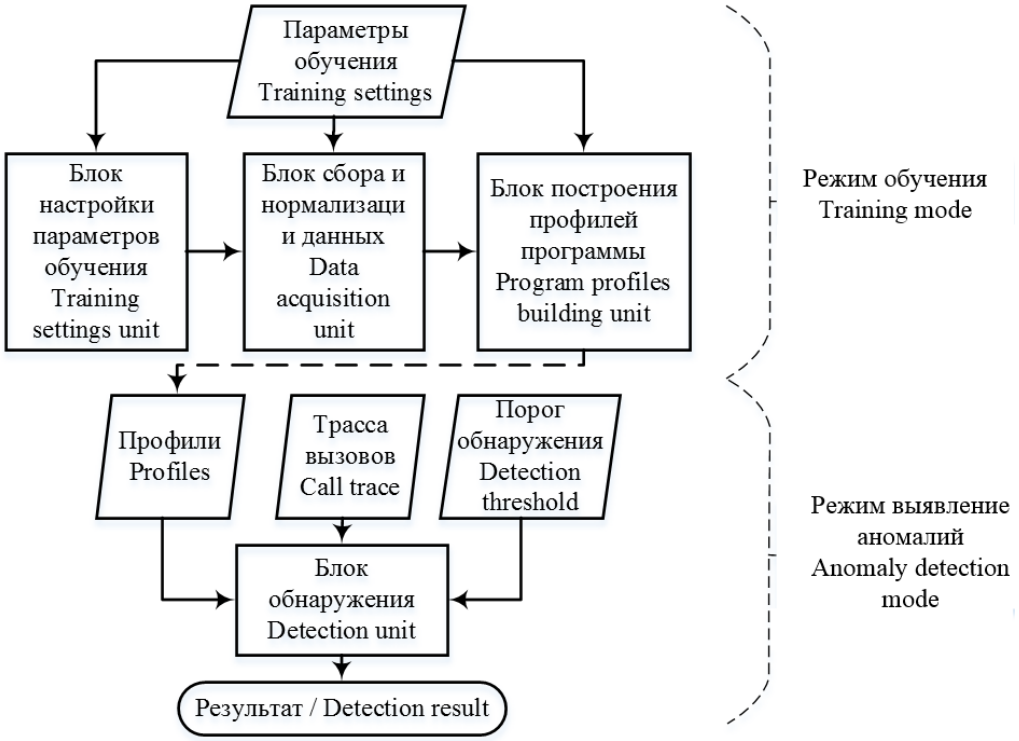


Fig. 1. Abnormal Behavior Detection System Diagram

Рис. 1. Схема системы обнаружения аномального поведения

Отметим, что чем больше мощность $\mathcal{I}(P)$, тем меньше вероятность принять нормальное поведение программы за аномалию. С другой стороны, увеличение $\mathcal{I}(P)$ приводит к увеличению периода обучения. В свою очередь увеличение $\mathcal{L}(P)$ может затруднить создателю эксплоита написание необнаруживаемого вредоносного кода. Однако увеличение числа отслеживаемых API-вызовов приводит к росту размера профиля и замедлению выполнения программы P. Актуальной задачей является выбор оптимальных наборов $\mathcal{I}(P)$ и $\mathcal{L}(P)$ (эта задача в настоящей работе не решается). Размер l цепочек в профиле $C_t(P, l)$ влияет на размер этого профиля. При малых значениях профиль компактный, но при этом возрастает вероятность пропуска вредоносного кода. При больших значениях l профиль растет и растет вероятность принять легитимную программу за вредоносную. В [2] экспериментальным путем установлено, что для операционных систем семейства Unix оптимальным значением l для обнаружения аномалий является 6.

Кратко опишем схему на рис. 1. В блоке сбора данных по выполняющейся программе P строится последовательность API-вызовов вида (1). Существуют различные способы мониторинга API-вызовов. Двумя наиболее распространенными способами является: 1) замена адресов импортируемых функций на адрес своего перехватчика и 2) замена начала вызываемых функций кодом перехода на свой обработчик [3]. В первом случае выполнение эксплоита может быть не обнаружено, так как эксплоит может вызывать функцию не через таблицу импорта, а передавать управление непосредственно на код самой API-функции. Поэтому для мониторинга следует использовать метод модификации начала импортируемых функций. Для мониторинга API-функций библиотек, загружаемых динамически, необходимо иметь возможность отслеживать загрузку библиотек. Средство мониторинга API-вызовов, реализующее блок сбора данных, должно позволять получать не только адрес ca API-вызова, но и адрес ba загрузки модуля, из которого выполняется этот вызов. Например, утилита API Monitoring [16] позволяет отслеживать для выбранного исполнимого файла

API-функции, вызываемые этим файлом, а также имеется возможность выбирать функции, вызов которых требуется отслеживать. В свою очередь средство ListDll [17] позволяет для активного процесса получить имена, адреса и размеры библиотек, загруженных в адресное пространство процесса. Таким образом, в совокупности эти средства позволяют построить последовательность троек вида (2), и поэтому может быть применено выражение (4). В блоке нормализации данные, накопленные в блоке сбора, проходят предварительную обработку. В частности, в этом блоке удаляются дубликаты API-вызовов. В блоке построения профиля формируются профили вида (5) и (6). Заметим, что для составления профиля (5) следует использовать трассы вызовов для разных потоков, и не использовать API-вызовы на границе переключения потоков, так как переключение потоков может быть в любой момент, и часто не зависит от логики программы. Построенные профили, вместе с трассой API-вызовов $\mathrm{Path}_{t_2}(P(I))$, $I \notin \mathcal{I}(P)$ затем используются как входные данные в блоке обнаружения, который реализует алгоритмы $CheckTrace$ (алгоритм 1) и $CheckTraceBack$ (алгоритм 2).

2. Эксперименты

Интернет-браузеры являются одними из наиболее часто используемых программ, что служит причиной большого внимания к этим программам со стороны создателей вредоносного кода и со стороны специалистов по защите [18, 19]. С целью исследования возможности применения разработанных алгоритмов в качестве защищаемой программы P выбран браузер FireFox. Для этого браузера в силу его популярности в открытом доступе имеются концептуальные эксплоиты [20—23], которые в настоящей работе используются в исследовательских целях.

Эксплоит, разработанный в работе [20], используя технику JIT-Spray, размещает исполнимый код (шеллкод) в памяти браузера FireFox версии 44.0.2 и передает на него управление при успешной эксплуатации уязвимости. В шеллкоде вызывается функция `WinExec` (из библиотеки `kernel32.dll`) для запуска калькулятора (приложение `calc.exe`). Отметим, что в приложении FireFox не предусмотрен вызов этой API-функции, поэтому функция `WinExec` не будет содержаться в профилях $\mathcal{D}_{t_1}(P)$ и $\mathcal{C}_{t_1}(P)$. Следовательно такой вызов будет расценен как аномалия в разработанных алгоритмах. Вообще говоря, вызов любой API-функции из шеллкода, не содержащейся в профилях $\mathcal{D}_{t_1}(P)$ и $\mathcal{C}_{t_1}(P)$, будет расценен как аномалия. К числу нетипичных API-вызовов для программы FireFox, кроме `WinExec`, относятся такие библиотечные API-функции как `ReadProcessMemory`, `WriteProcessMemory`, `CreateProcess` (отметим, что эти функции могут использоваться ВПО [3]). Поэтому интерес представляют шеллкоды, в которых вызываются API-функции, являющиеся типичными для трассы вызовов при легитимном выполнении приложения P. Как показано в [3], к числу типичных для ВПО функций относятся функции из таб. 2, которые, как также являются типичными и для программы FireFox. С этой целью в настоящей работе созданы модификации шеллкода из [20], в каждой из которых вызывается API-функция из таб. 2. Таким образом, $N^E = 15$. В качестве отслеживаемых API-функций, то есть тех функций, из вызовов которых формируется трасса (1), выбраны также все функции из таблицы 2. Входными данными для браузера FireFox выбраны сайты пятнадцати различных тематик в соответствии с классификацией, используемой на сайте `www.similarweb.com`: 1) новости, 2) сайты государственных учреждений, 3) искусство и развлечения, 4) бизнес, 5) сайты сообществ/социальных сетей, 6) технологии, 7) электронная коммерция, 8) финансовые сервисы, 9) продукты/еда, 10) игровые сайты, 11) хобби, 12) поиск работы 14) устройства/сенсоры, 15) путешествия. Список сайтов для каждой группы (тематики) формировался с помощью сервиса `www.similarweb.com`. Для исследования зависимости вероятности ложного обнаружения от размера обучающей выборки сформировано случайным образом 15 выборок размера от 10 до 150 сайтов с шагом 10. Для формирования каждой выборки использовались первые 10 наиболее посещаемых сайтов каждой группы[1]; каждая последующая выборка содержит предыдущую

[1] по состоянию на 20.02.2020

Table 2. Examples of the API functions typical for the FireFox and typical for the malware

Таблица 2. Примеры API-функций, типичных для программы FireFox и типичных для вредоносного ПО

Вредоносная активность/ Malicious activity	**API**	**DLL**
Клавиатурный шпион/ Key Logger	`FindWindowW/FindWindowExW`, `SetWindowsHookExA/SetWindowsHookExW`	`user32.dll`
Захват экрана/ Screen Capture	`GetDC`, `GetWindowDC`	`user32.dll`
	`CreateCompatibleDC`, `CreateCompatibleBitmap`	`gdi32.dll`
	`WriteFile/WriteFileEx`	`kernel32.dll`
Противодействие отладке/ Antidebugging	`IsDebuggerPresent`	
Внедрение кода/ DLL Injection	`VirtualAlloc/VirtualAllocEx`	
Инсталлятор/ Dropper	`LoadResource`, `SizeOfResource`	

(меньшего размера). В качестве тестовых наборов сформировано случайным образом 30 наборов по 15 сайтов в каждом наборе. Для тестовых наборов использовались первые 30 сайтов каждой тематики. Таким образом обучающая выборка и тестовая выборка пересекаются, что характеризует ситуацию, когда пользователь при обучении и при тестировании может посещать одинаковые сайты. Результаты оценки вероятности ложного обнаружения P_{FP}, усредненные по 30 тестовым выборкам, максимальная $l^{t_1}_{max}(P)$ и средняя $\mathrm{E}[l^{t_1}(P)]$ длина списка расстояний, а также размер S профиля расстояний (в байтах) приведены в таблице 3. Отметим, что при проведении экспериментов в алгоритмах $CheckTrace$ и $CheckTraceBack$ значение l выбрано равным двум, с целью оценить возможность обнаружения аномалий только на основе профиля расстояний. В качестве порога T выбраны значения 30 и 60.

Как видно из таблицы, с ростом размера обучающей выборки вероятность ложного обнаружения для обоих алгоритмов уменьшается. При этом для алгоритма $CheckTraceBack$ вероятность ложного обнаружения приближается к нулю уже при 150-ти сайтах в обучающей выборке. Из таблицы также видно, что резкое уменьшение вероятности ложного обнаружения при переходе от выборки размера 100 к выборке размера 110 сопровождается ростом размера базы (столбец S). Это может свидетельствовать о том, что на этапе обучения в выборке из 110 сайтов были такие сайты, которые в большей степени задействовали функционал программы P, чем сайты из предыдущих выборок. Для всех модификаций эксплоита оба алгоритма показали стопроцентное обнаружение при всех возможных значениях $|\mathcal{I}(P)|$.

3. Заключение

К недостаткам разработанного способ, кроме отмеченных в разделе 1.1, следует отнести необходимость запуска приложения, что может привести к запуску эксплоита. Тем не менее полагается, что позднее обнаружение эксплоита предпочтительнее его пропуска. Кроме того, алгоритмы могут быть адаптированы для динамического обнаружения эксплоитов на ранней стадии запуска, при выявлении нетипичного расстояния между функциями: если проверка нетипичности вызова осуществляется до выполнения вызываемой функции. Однако в этом случае возможно замедле-

Table 3. Dependence of the $CheckTrace$ and $CheckTraceBack$ algorithms characteristics on the training sample size for the FireFox

Таблица 3. Зависимость характеристик алгоритмов $CheckTrace$ и $CheckTraceBack$ от размера обучающей выборки для программы FireFox

$\lvert\mathcal{I}(P)\rvert$	P_{FP}				$l^{t_1}_{max}(P)$	$\mathrm{E}[l^{t_1}(P)]$	S
	$CheckTrace$		$CheckTraceBack$				
	$T = 30$	$T = 60$	$T = 30$	$T = 60$			
10	1	1	1	1	27	4	4634
20	1	1	0,38125	0,28	31	4	5499
30	1	1	0,35625	0,27	32	5	5742
40	1	1	0,3125	0,228	32	5	5910
50	1	0,946	0,3125	0,212	33	5	6010
60	1	0,826	0,3125	0,21	34	5	6140
70	0,99375	0,786	0,275	0,186	34	5	6254
80	0,84375	0,672	0,2375	0,164	35	5	6439
90	0,725	0,592	0,2375	0,162	37	6	6676
100	0,61875	0,506	0,2375	0,162	38	6	6977
110	0,36875	0,322	0,03125	0,012	38	6	7450
120	0,29375	0,294	0,03125	0,012	43	6	7658
130	0,2875	0,274	0,03125	0,01	44	6	7780
140	0,2375	0,218	0,03125	0,01	44	6	8095
150	0,21875	0,21	0,01875	0,004	45	6	8274

ние выполнения защищаемой программы P. К недостаткам следует отнести также необходимость переобучения системы защиты при обновлении защищаемого программного обеспечения.

Не смотря на отмеченные недостатки, рассмотренный подход на основе подсчета расстояний между соседними вызовами функций может использоваться при исследовании недоверенных входных данных на наличие в них эксплоита. Для оценки вероятности ложного пропуска P_{TP}, а также для уточнения оптимального значения T требуется проведение исследования для разных типов известных эксплоитов.

References

[1] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, “Survey of intrusion detection systems: techniques, datasets and challenges”, *Cybersecurity*, vol. 2, no. 1, p. 20, 2019.

[2] S. Forrest, S. Hofmeyr, and A. Somayaji, “The Evolution of System-Call Monitoring”, in *Proceedings of 2008 Annual Computer Security Applications Conference (ACSAC)*, 2008, pp. 418–430.

[3] S. Gupta, H. Sharma, and S. Kaur, “Malware Characterization Using Windows API Call Sequences”, *Journal of Cyber Security and Mobility*, vol. 7, no. 4, pp. 363–378, 2018.

[4] R. Veeramani and N. Rai, “Windows API based Malware Detection and Framework Analysis”, *International Journal of Scientific & Engineering Research*, vol. 3, no. 3, pp. 1–6, 2012.

[5] A. Singh, R. Arora, and H. Pareek, “Malware Analysis using Multiple API Sequence Mining Control Flow Graph”, *CoRR. arXiv preprint arXiv:1707.02691*, 2017.

[6] M. L. Bernardi, M. Cimitile, D. Distante, F. Martinelli, and F. Mercaldo, “Dynamic malware detection and phylogeny analysis using process mining”, *International Journal of Information Security*, vol. 18, no. 3, pp. 257–284, 2019.

[7] L. Viljanen, "A Survey of Application Level Intrusion Detection", *Technical report, Series of Publications C, Report C-2004-61 Helsinki*, 2004.

[8] G. Creech, "Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks", PhD thesis, University of New South Wales, Canberra, Australia, 2014.

[9] H. Hu, S. Shinde, S. Adrian, Z. L. Chua, P. Saxena, and Z. Liang, "Data-oriented programming: On the expressiveness of non-control data attacks", in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 969–986.

[10] K. K. Ispoglou, B. AlBassam, T. Jaeger, and M. Payer, "Block Oriented Programming: Automating Data-Only Attacks", in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 1868–1882.

[11] Y. V. Kosolapov, "About detection of code reuse attacks", *Modelirovanie i Analiz Informatsionnykh Sistem*, vol. 26, no. 2, pp. 213–228, 2019.

[12] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems", in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, 2002, pp. 255–264.

[13] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A.-R. Sadeghi, "Just-In-Time Code Reuse: On the Effectiveness of Fine-Grained Address Space Layout Randomization", in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 574–588.

[14] E. Stalmans and S. El-Sherei, *Macro-less Code Exec in MSWord*, Last access 12.12.2019. [Online]. Available: https://sensepost.com/blog/2017/macro-less-code-exec-in-msword/.

[15] P. D. Borisov and Y. V. Kosolapov, "On the automatic analysis of the practical resistance of obfusting transformations", *Modelirovanie i Analiz Informatsionnykh Sistem*, vol. 26, no. 3, pp. 317–331, 2019.

[16] *API Monito*, Last access 28.11.2019. [Online]. Available: http://www.rohitab.com/apimonitor.

[17] *ListDLLs*, Last access 28.11.2019. [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/listdlls.

[18] M. Vervier, M. Orru, B. J. Wever, and E. Sesterhenn, *Browser Security Whitepaper*, Last access 05.12.2019. [Online]. Available: https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf.

[19] R. Gawlik and T. Holz, "Sok: Make JIT-spray great again", in *WOOT'18 Proceedings of the 12th USENIX Conference on Offensive Technologies*, 2018, pp. 1–14.

[20] *Offensive Security, Exploitdb/exploits/windows/remote/42484.html*, Last access 05.12.2019. [Online]. Available: https://github.com/offensive-security/exploitdb/blob/master/exploits/windows/remote/42484.html.

[21] 0vercl0k, *CVE-2019-9810*, Last access 05.12.2019. [Online]. Available: https://github.com/0vercl0k/CVE-2019-9810.

[22] *Exploit Database*, Last access 05.12.2019. [Online]. Available: https://www.exploit-db.com/.

[23] *CVE-2017-5375_ASM.JS_JIT-Spray*, Last access 30.12.2019. [Online]. Available: https://github.com/rh0dev/expdev/tree/master.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 27, NO. 2, 2020
journal homepage: www.mais-journal.ru

COMPUTER SYSTEM ORGANIZATION

On a Mechanism for the Formation of Spatially Inhomogeneous Structures of Light Waves in Optical Information Transmission Systems

E. P. Kubyshkin[1], V. A. Kulikov[1]

DOI: 10.18255/1818-1015-2020-2-152-163

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

MSC2020: 37G15, 78A40
Research article
Full text in Russian

Received June 1, 2020
After revision June 10, 2020
Accepted June 10, 2020

Spatially inhomogeneous structures of light waves are used as a mechanism of compacting information in optical and fiber-optic communication systems. In this paper, we consider a mathematical model of an optical radiation generator with a nonlinear delayed feedback loop and a stretching (compression) operator of the spatial coordinates of the light wave in a plane orthogonal to the radiation direction. It is shown that the presence of a delay in the feedback loop can lead to the generation of stable periodic spatially inhomogeneous oscillations. In the space of the main parameters of the generator, the spaces of generation of stable spatially non-uniform oscillations are constructed, the mechanism of their occurrence is studied, and approximate asymptotic formulas are constructed.

Keywords: spatially inhomogeneous waves, bifurcation, optical information transmission systems.

INFORMATION ABOUT THE AUTHORS

Evgenii P. Kubyshkin correspondence author	orcid.org/0000-0003-1796-0190. E-mail: kubysh.e@yandex.ru Doctor of Science, Professor.
Vladimir A. Kulikov	orcid.org/0000-0003-1351-7706. E-mail: kulikov7677@gmail.com Graduate student.

Funding: RFBR, project No 19-31-90133.

For citation: E. P. Kubyshkin and V. A. Kulikov, “On a Mechanism for the Formation of Spatially Inhomogeneous Structures of Light Waves in Optical Information Transmission Systems”, *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 152-163, 2020.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 27, № 2, 2020
сайт журнала: www.mais-journal.ru

COMPUTER SYSTEM ORGANIZATION

Об одном механизме образования пространственно-неоднородных структур световых волн в оптических системах передачи информации

Е. П. Кубышкин[1], В. А. Куликов[1]

DOI: 10.18255/1818-1015-2020-2-152-163

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

УДК 517.9, 535.015
Научная статья
Полный текст на русском языке

Получена 1 июня 2020 г.
После доработки 10 июня 2020 г.
Принята к публикации 10 июня 2020 г.

Пространственно-неоднородные структуры световых волн используются как механизм уплотнения информации в системах оптической и волоконно-оптической связи. В работе рассматривается математическая модель генератора оптического излучения с контуром нелинейной запаздывающей обратной связи и оператором растяжения (сжатия) пространственных координат световой волны в плоскости, ортогональной направлению излучения. Показано, что наличие запаздывания в контуре обратной связи может привести к генерации устойчивых периодических пространственно-неоднородных колебаний. В пространстве основных параметров генератора построены области генерации устойчивых пространственно-неоднородных колебаний, изучен механизм их возникновения, построены приближенные асимптотические формулы.

Ключевые слова: пространственно-неоднородные волны, бифуркация, оптические системы передачи информации.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Евгений Павлович Кубышкин автор для корреспонденции	orcid.org/0000-0003-1796-0190. E-mail: kubysh.e@yandex.ru д-р. физ.-мат. наук, профессор.
Владимир Александрович Куликов	orcid.org/0000-0003-1351-7706. E-mail: kulikov7677@gmail.com аспирант.

Финансирование: РФФИ, проект № 19-31-90133.

Для цитирования: E. P. Kubyshkin and V. A. Kulikov, “On a Mechanism for the Formation of Spatially Inhomogeneous Structures of Light Waves in Optical Information Transmission Systems”, *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 152-163, 2020.

Введение

В работе [1—3] приведены экспериментальные результаты образования пространственно-неоднородных волн в лазерных пучках генератора оптического излучения со специальным нелинейным контуром двумерной обратной связи. Такие структуры возникают в плоскости, ортогональной направлению распространения световой волны. Их возникновение обусловлено нелинейностью системы, которая обеспечивается тонким слоем нелинейной проводящей среды и контуром двумерной обратной связи с оператором пространственного преобразования световой волны в плоскости излучения оптического генератора. В [1] также предложена математическая модель для описания этого явления и приведены результаты ее численного анализа в случае оператора поворота плоскости световой волны. Математическая модель представляет собой начально-краевую задачу для нелинейного дифференциального уравнения параболического типа с оператором преобразования пространственного аргумента, которое рассматривается в области, определяемой апертурой светового излучения. Эта начально-краевая задача и различные ее обобщения изучались в большом количестве работ, где различными аналитическими и численными методами строятся пространственно-неоднородные решения. Обзор данных публикаций приведен, например, в [4]. В работах в основном рассматривается оператор поворота пространственного аргумента, так как такой оператор создается механизмом генерации автоколебательных решений. Модель работы [1] не учитывает фактор временного запаздывания в контуре нелинейной обратной связи. Временное запаздывание также может служить механизмом возбуждения автоколебаний [4, 5], в том числе при более простом, с конструктивной точки зрения, операторе преобразования пространственных координат. В настоящей работе рассматривается математическая модель работы [1] в круговой области с оператором растяжения (сжатия) пространственного аргумента и временным запаздыванием в контуре обратной связи, для которой исследуются условия и характер потери устойчивости состояний равновесия в зависимости от коэффициента усиления и величины запаздывания. Показана возможность колебательной потери устойчивости состояний равновесия и возможность бифуркации устойчивых пространственно-неоднородных периодических решений. Такие решения могут быть использованы как носители информации в оптических и волоконно-оптических системах связи.

1. Математическая постановка задачи

Для функционально-дифференциального уравнения с запаздывающим аргументом

$$u_t(\rho,\phi,t) + u(\rho,\phi,t) = D\Delta_{\rho\phi}u(\rho,\phi,t) + K(1+\gamma cos(Q_\alpha u(\rho,\phi,t-T))) \tag{1}$$

относительно функции $u(\rho,\phi,t+s)$, заданной в полярных координатах $0 \le \rho \le R, 0 \le \phi \le 2\pi$ $(R>0)$ и $t \ge 0, -T \le s \le 0$ $(T>0)$, в котором $\Delta_{\rho\phi}$ – оператор Лапласа в полярных координатах, Q_α – оператор преобразования пространственных координат, D, K – положительные постоянные, $0 < \gamma < 1$, в области $\bar{K}_R \times \mathbb{R}^+$, где круг $\bar{K}_R = \{(\rho,\phi) : 0 \le \rho \le R, 0 \le \phi \le 2\pi\}$, $\mathbb{R}^+ = \{t : 0 \le t < \infty\}$, рассматривается начально-краевая задача вида

$$u_\rho(R,\phi,t) = 0, \quad u(\rho,0,t) = u(\rho,2\pi,t), \quad u_\phi(\rho,0,t) = u_\phi(\rho,2\pi,t),$$

$$u(\rho,\phi,t+s)|_{t=0} = u_0(\rho,\phi,s) \in H_0(K_R;-T,0). \tag{2}$$

В (1) при $0 < \alpha < 1$

$$Q_\alpha u(\rho,\phi,t) = \begin{cases} \alpha^{-2}u(\rho/\alpha,\phi,t), & 0 \le \rho \le \alpha, \\ 0, & \alpha < \rho \le 1, \end{cases} \tag{3}$$

– оператор сжатия пространственных координат, при $\alpha > 1$

$$Q_\alpha u(\rho,\phi,t) = \alpha^{-2}u(\rho/\alpha,\phi,t), \ 0 \le \rho \le 1, \tag{4}$$

– оператор растяжения пространственных координат; в (2) пространство начальных условий

$$H_0(K_R;-T,0)=\{u(\rho,\phi,s)\ :\ u(\rho,\phi,s)\in C(\bar{K}_R\times[-T,0]),u(\rho,0,s)=u(\rho,2\pi,s),u_\phi(\rho,0,s)=u_\phi(\rho,2\pi,s),$$

при каждом s $u(\rho,\phi,s)\in H^2(K_R)\}$,

где пространство функций $H^2(K_R)\subset W_2^2(K_R)$ и получено замыканием множества функций

$$\{u(\rho,\phi)\ :\ u(\rho,\phi)\in C^2(\bar{K}_R),u_\rho(R,\phi)=0,u(\rho,0)=u(\rho,2\pi),u_\phi(\rho,0)=u_\phi(\rho,2\pi)\}$$

в метрике пространства функций $W_2^2(K_R)$. В дальнейшем $L_2(K_R)$ – пространство вещественнозначных определенных в K_R функций $u(\rho,\phi)$, для которых

$$||u(\rho,\phi)||_{L_2}=(u(\rho,\phi),u(\rho,\phi))_{L_2}^{1/2}<\infty,\ (u(\rho,\phi),v(\rho,\phi))_{L_2}=\int_{K_R}\rho u(\rho,\phi)v(\rho,\phi)d\rho d\phi,$$

здесь и в дальнейшем

$$W_2^2(K_R)\subset L_2(K_R),W_2^2(K_R)=\{u(\rho,\phi)\ :\ ||u(\rho,\phi)||_{W_2^2}=(u(\rho,\phi),u(\rho,\phi))_{W_2^2}^{1/2}<\infty,(u(\rho,\phi),v(\rho,\phi))_{W_2^2}=$$
$$(u(\rho,\phi),v(\rho,\phi))_{L_2}+(\Delta_{\rho\phi}u(\rho,\phi),\Delta_{\rho\phi}v(\rho,\phi))_{L_2},$$

$C(\bar{K}_R)$ и $C^2(\bar{K}_R)$ пространства непрерывных и дважды непрерывно дифференцируемых в $\bar{K}_R$ функций, для которых определена норма

$$||u(\rho,\phi)||_C=\max_{\rho,\phi}|u(\rho,\phi)|,\ ||u(\rho,\phi)||_{C^2}=||u(\rho,\phi)||_C+||\Delta_{\rho\phi}u(\rho,\phi)||_C<\infty.$$

Фазовым пространством начально-краевой задачи (1)-(2) является пространство

$$H(K_R;-T,0)=\{u(\rho,\phi,s)\ :\ u(\rho,\phi,s)\in L_2(K_R)$$

при каждом $-T\le s\le 0,||u(\rho,\phi,s)||_{L_2}\in C([-T,0])\}$,

норму в котором определим как

$$||u(\rho,\phi,s)||_H=\max_s||u(\rho,\phi,s)||_{L_2}.$$

Областью определения правой части уравнения (1) является пространство $H_0(K_R;-T,0)$. Норму в $H_0(K_R;-T,0)$ определим как

$$||u(\rho,\phi,s)||_{H_0}=\max_s||u(\rho,\phi,s)||_{W_2^2}.$$

Под решением начально-краевой задачи (1)-(2), определенным при $t>0$, будем понимать функцию $u(\rho,\phi,t+s)\in H_0(K_R;-T,0)$ (при каждом $t>0$), непрерывно дифференцируемую по t при $t>0$, обращающую уравнение (1) в тождество в фазовом пространстве и удовлетворяющую начальным условиям (2).

В работе изучаются условия и характер потери устойчивости состояний равновесия $u_*(\rho,K,\gamma)$ и обусловленные ею бифуркации пространственно-неоднородных автоколебательных решений начально-краевой задачи (1)-(2), а также их устойчивость.

2. Анализ устойчивости состояний равновесия начально-краевой задачи (1)-(2)

Состояния равновесия начально-краевой задачи (1)-(2) определяются решениями $u_*(\rho) = u_*(\rho, K, \gamma) \in H^2(K_R)$ нелинейного операторного уравнения

$$u(\rho, \phi) = D\Delta_{\rho\phi}u(\rho, \phi) + K(1 + \gamma cos(Q_\alpha u(\rho, \phi))) \tag{5}$$

в $L_2(K_R)$. Выберем одно из решений $u_*(\rho) = u_*(\rho, K, \gamma)$ уравнения (5) и запишем начально-краевую задачу (1)-(2) в его окрестности, заменив $u(\rho, \phi, t) \longrightarrow u_*(\rho, K, \gamma) + u(\rho, \phi, t)$. В результате получим начально-краевую задачу

$$u_t(\rho, \phi, t) + u(\rho, \phi, t) = D\Delta_{\rho\phi}u(\rho, \phi, t) - b(\rho)Q_\alpha u(\rho, \phi, t - T) + \\ + b_2(\rho)(Q_\alpha u(\rho, \phi, t - T))^2/2 + b(\rho)(Q_\alpha u(\rho, \phi, t - T))^3/6 + \dots, \tag{6}$$

$$u_\rho(R, \phi, t) = 0, \quad u(\rho, 0, t) = u(\rho, 2\pi, t), \quad u_\phi(\rho, 0, t) = u_\phi(\rho, 2\pi, t),$$

$$u(\rho, \phi, s) = u_0(\rho, \phi, s) \in H_0(K_R; -T, 0), \tag{7}$$

$$b(\rho) = K\gamma \sin(Q_\alpha u_*(\rho, K, \gamma)), \quad b_2(\rho) = -K\gamma \cos(Q_\alpha u_*(\rho, K, \gamma)), \tag{8}$$

где точками обозначены слагаемые, имеющие по $Q_\alpha u(\rho, \phi, t - T)$ более высокий порядок малости в норме $L_2(K_R)$.

Рассмотрим линейную часть (6)-(7)

$$u_t(\rho, \phi, t) + u(\rho, \phi, t) = D\Delta_{\rho\phi}u(\rho, \phi, t) - b(\rho)Q_\alpha u(\rho, \phi, t - T), \tag{9}$$

$$u_\rho(R, \phi, t) = 0, \quad u(\rho, 0, t) = u(\rho, 2\pi, t), \quad u_\phi(\rho, 0, t) = u_\phi(\rho, 2\pi, t),$$

$$u(\rho, \phi, s) = u_0(\rho, \phi, s) \in H_0(K_R; -T, 0). \tag{10}$$

Определяя решения (9)-(10) вида $u(\rho, \phi, t) = u(\rho, \phi)e^{\lambda t}, \lambda \in \mathbb{C}$ (решения Эйлера) получим пучок операторов

$$P(\lambda)u(\rho, \phi) \equiv \lambda u(\rho, \phi) + u(\rho, \phi) - D\Delta_{\rho,\phi}u(\rho, \phi) + b(\rho)Q_\alpha u(\rho, \phi)e^{-\lambda T}, \tag{11}$$

действующий в $\tilde{L}_2(K_R)$ с областью определения $\tilde{H}^2(K_R)$, точки спектра которого определяют устойчивость решений начально-краевой задачи (9)-(10), а соответствующие им собственные функции решения искомого вида. Здесь и в дальнейшем знаком “тильде” будем обозначать комплексное расширение соответствующего функционального пространства, скалярное произведение и норма в котором обобщается стандартным образом.

Представим $u(\rho, \phi) \in H^2(K_R)$ в виде

$$u(\rho, \phi) = u_0(\rho)v_{00} + \sum_{n=-\infty}^{\infty}\sum_{j=1}^{\infty} u_j(\rho, n\phi)v_{nj}, \quad u_0(\rho) = \frac{1}{\sqrt{\pi}R}, \quad u_j(\rho, n\phi) = \frac{R_{nj}(\rho)e^{in\phi}}{(2\pi)^{1/2}},$$

$$R_{nj}(\rho) = \frac{\sqrt{2}/RJ_n(\gamma_{nj}\rho/R)}{(1 - n^2/\gamma_{jn}^2)^{1/2}|J_n(\gamma_{nj})|} \ (n \geq 0), \ R_{-nj}(\rho) = R_{nj}(\rho), \ u_j(\rho, 0) \equiv u_j(\rho),$$

$$\int_0^R \rho R_{nj}(\rho)R_{np}(\rho)d\rho = \delta_{jp}, i = \sqrt{-1}, v_{00}, v_{0j} \in \mathbb{R}, v_{nj} \in \mathbb{C}, v_{-nj} = \bar{v}_{nj}, \tag{12}$$

где $J_n(\rho)$ функции Бесселя первого рода n - го порядка, γ_{nj} j -й положительный ноль функции $J_n'(\rho)$, $\gamma_{00} = 0$, δ_{jp} – символ Кронекера, $v_0 = (v_{00}, v_{01}, v_{01}, \dots) \in l_2^2 \subset l_2, l_2^2 = \{v = (v_0, v_1, v_2, \dots), v_k \in \mathbb{R} : \|v\|_{l_2^2} = (v_0^2 + \sum_{k=1}^{\infty} k^4 v_k^2)^{1/2} < \infty\}$, $v_n = (0, v_{n1}, v_{n2}, v_{n3}, \dots) \in \tilde{l}_2^2 \subset \tilde{l}_2, \tilde{l}_2^2 = \{(v_0, v_1, v_2, v_3, \dots), v_k \in \mathbb{C} : \|v\|_{\tilde{l}_2^2} = (|v_0|^2 +$

$\sum_{k=1}^{\infty} k^4|v_k|^2)^{1/2} < \infty\}$. Функции $u_j(\rho, n\phi)$, являясь полной системой собственных функций оператора Лапласа, образуют ортогональный базис в $\tilde{H}^2(K_R)$ и ортонормированный в $\tilde{L}_2(K_R)$, т.е. $v_0 \in l_2^2$ и $v_n \in \tilde{l}_2^2$ определяются однозначно. Подставим ряд (12) в (11) и спроектируем на $u_0(\rho)$, $u_j(\rho, n\phi)$, $j, n = 1, 2, \dots$. В результате получим последовательность операторных уравнений в пространстве $\tilde{l}_2$ с областью определения $\tilde{l}_2^2$ вида

$$P^{(n)}(\lambda, \alpha)v_n = 0, \quad n = 0, 1, \dots, \tag{13}$$

для определения $v_n \in \tilde{l}_2^2$, где $P^{(n)}(\lambda, \alpha)$ бесконечномерные матрицы с элементами

$$P_{jj}^{(n)}(\lambda, \alpha) = \lambda + 1 + D\gamma_{nj}^2 + p_{jj}^{(n)}(\alpha)e^{-\lambda T}, \quad P_{jq}^{(n)}(\lambda, \alpha) = p_{jq}^{(n)}(\alpha)e^{-\lambda T}, \; j \neq q,$$

$$p_{jq}^{(n)}(\alpha) = \alpha^{-2}\int_0^{r_\alpha} \rho b(\rho)R_{nq}(\rho/\alpha)R_{np}(\rho)d\rho, \quad r_\alpha = \alpha \; (\alpha < 1), \quad r_\alpha = 1 \; (\alpha > 1), \tag{14}$$

$j, q = 0, 1, \dots$, при $n = 0$, и $j, q = 1, 2, \dots$, при $n > 0$. Отметим, что коэффициенты $p_{jq}^{(n)}(\alpha) \longrightarrow 0$ при $j, q \longrightarrow \infty$.

Совокупность значений $\lambda_*^{(n)}$, при которых операторные уравнения (13) имеют ненулевые решения $v_*^{(n)} \in \tilde{l}_2^2$ определяет множество точек спектра пучка операторов (11), а решения $v_*^{(n)}$ с учетом (12) соответствующие собственные функции. Анализ расположения $\lambda_*^{(n)}$ позволяет построить в пространстве параметров области устойчивости решений начально-краевой задачи (9)-(10).

Рассмотрим оператор растяжения (4). В этом случае начально-краевая задача (1)-(2) может иметь однородные состояния равновесия $u_*(\rho, K, \gamma) \equiv u_*(K, \gamma)$, Однородные состояния равновесия $u_* = u_*(K, \gamma)$ начально-краевой задачи (1)-(2) определяются как решения уравнения

$$u_* = K(1 + \gamma cos u_*). \tag{15}$$

Уравнение (15) в зависимости от K и γ может иметь несколько решений, в том числе кратные. Отметим, что $b(\rho) \equiv b = K\gamma \sin(u_*(K, \gamma))$. С учетом этого в (13), (14) $p_{jq}^{(n)}(\alpha)$ представим в виде $bp_{jq}^{(n)}(\alpha)$. Исследуем условия потери устойчивости состоянием равновесия $u_*(K, \gamma)$. Воспользуемся для этого методом D-разбиения [6]. Положим в (13) $\lambda = i\omega$, $\omega \geq 0$ и рассмотрим для каждого n последовательность "усеченных" конечномерных матриц $P_m^{(n)}(i\omega, \alpha)$, в которых $j, q = 0, 1, ..., m$ при $n = 0$, и $j, q = 1, 2, ..., m$ при $n > 0$. Рассмотрим сначала случай $n = 0$. Приравняем нулю определитель матрицы $P_m^{(0)}(i\omega, \alpha)$ и выразим из этого равенства элемент $P_{jj}^{(0)}(i\omega, \alpha)$. В результате получим выражение

$$i\omega + 1 + D\gamma_{0j}^2 + bp_{jj}^{(0)}(\alpha)e^{-i\omega T} + \Delta_{m1}^{(j)}(b, T, \omega, \alpha) + i\Delta_{m2}^{(j)}(b, T, \omega, \alpha) = 0, \tag{16}$$

в котором функции $\Delta_{m1}^{(j)}(b, T, \omega, \alpha)$ и $\Delta_{m2}^{(j)}(b, T, \omega, \alpha)$ получены в результате объединения выражений, не содержащих $P_{jj}^{(0)}(i\omega, \alpha)$. Выделив в (16) вещественную и мнимую часть, получим равенства

$$1 + D\gamma_{0j}^2 + bp_{jj}^{(0)}(\alpha)\cos(\omega T) + \Delta_{m1}^{(j)}(b, T, \omega, \alpha) = 0, \quad \omega - bp_{jj}^{(0)}(\alpha)\sin(\omega T) + \Delta_{m2}^{(j)}(b, T, \omega, \alpha) = 0,$$

из которых находим

$$b = b(\omega) = (-1)^{k+1}(1 + D\gamma_{0j}^2 + \Delta_{m1}^{(j)}(b, T, \omega, \alpha))/(p_{jj}^{(0)}(\alpha)\cos(\text{arctg}((\omega + \Delta_{m2}^{(j)}(b, T, \omega, \alpha))/$$

$$(1 + D\gamma_{0j}^2 + \Delta_{m1}^{(j)}(b, T, \omega, \alpha))))), \tag{17}$$

$$T = T(\omega) = \omega^{-1}(\pi k - \text{arctg}(\omega + \Delta_{m2}^{(j)}(b, T, \omega, \alpha))/(1 + D\gamma_{0j}^2 + \Delta_{m1}^{(j)}(b, T, \omega, \alpha))), \quad k = 1, 2, \dots. \tag{18}$$

Аналогичные выражения могут быть получены и для других n. В случае $\alpha = 1$ функции $\Delta^{(j)}_{m1}(b, T, \omega, \alpha) \equiv 0$ и $\Delta^{(j)}_{m2}(b, T, \omega, \alpha) \equiv 0$. В связи с этим выражения (17), (18) определяют совокупность кривых в плоскости (b, T) на которых пучек операторов (11) имеет точки спектра, расположенные на мнимой оси комплексной плоскости. Это дает возможность построить в плоскости (b, T) (при фиксированных других параметрах) область устойчивости решений начально-краевой задачи (9)-(10). В случае $\alpha > 1$ по (17), (18) построим итерационный процесс

$$b^{(q)}(\omega) = (-1)^{k+1}(1 + D\gamma_{0j}^2 + \Delta^{(j)}_{m1}(b^{(q-1)}(\omega), T^{(q-1)}(\omega), \omega, \alpha))/(p^{(0)}_{jj}(\alpha)\cos(\operatorname{arctg}((\omega +$$

$$+ \Delta^{(j)}_{m2}(b^{(q-1)}(\omega), T^{(q-1)}(\omega), \omega, \alpha))/(1 + D\gamma_{0j}^2 + \Delta^{(j)}_{m1}(b^{(q-1)}(\omega), T^{(q-1)}(\omega), \omega, \alpha))))), \tag{19}$$

$$T^{(q)}(\omega) = \omega^{-1}(\pi k - \operatorname{arctg}(\omega + \Delta^{(j)}_{m2}(b^{(q-1)}(\omega), T^{(q-1)}(\omega), \omega, \alpha))/$$

$$/(1 + D\gamma_{0j}^2 + \Delta^{(j)}_{m1}(b^{(q-1)}(\omega), T^{(q-1)}(\omega), \omega, \alpha))), \quad k = 1, 2, \ldots, \quad q = 1, 2, \ldots. \tag{20}$$

В качестве $b^{(0)}(\omega)$, $T^{(0)}(\omega)$ выбираем решение (17), (18) при $\Delta^{(j)}_{m1}(b, T, \omega, \alpha) \equiv 0$, $\Delta^{(j)}_{m2}(b, T, \omega, \alpha) \equiv 0$. Итерационный процесс (19), (20) достаточно быстро сходится как по q, так и по m, в силу достаточно быстрого стремления к нулю коэффициентов $p^{(n)}_{jq}(\alpha)$ матриц (13).

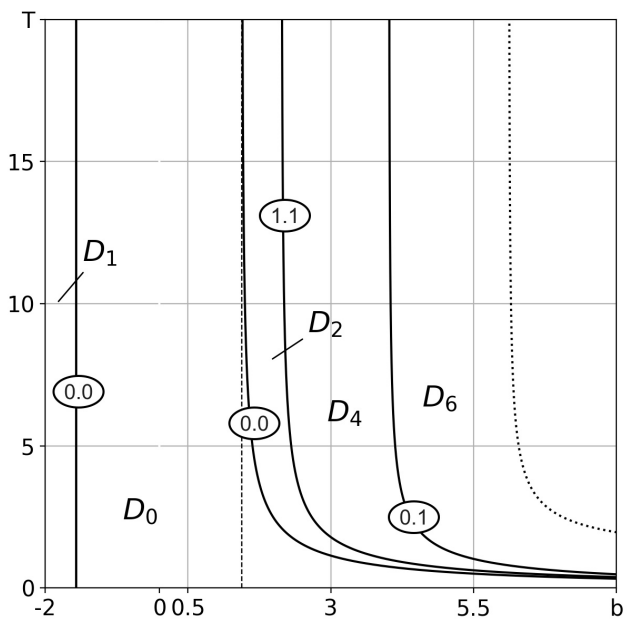


Fig. 1. The D-partition pattern of the plane (b, T) for $\alpha = 1.2$

Рис. 1. Картина D-разбиения плоскости (b, T) для $\alpha = 1.2$

На рисунках 1 и 2 для значений $D = 0.1$, $\gamma = 0.5$, $\alpha = 1.2$ и $\alpha = 1.4$ приведена картина D разбиения плоскости (b, T). На рисунках через D_j обозначены области, при значении параметров из которых пучок операторов (11) имеет j точек спектра, принадлежащих правой комплексной полуплоскости, а границы этих областей соответствуют точкам спектра, лежащим на мнимой оси. Из рисунков видно, что имеются значения параметров на границе области устойчивости (область

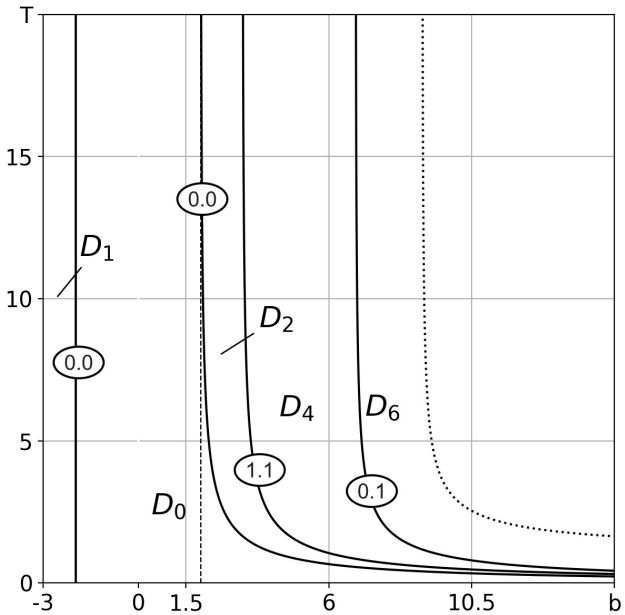


Fig. 2. The D-partition pattern of the plane (b, T) for $\alpha = 1.4$

Рис. 2. Картина D-разбиения плоскости (b, T) для $\alpha = 1.4$

D_0), при которых в правую полуплоскость может переходить пара комплексно сопряженных точек спектра (11) (правая граница), а также значения параметров, соответствующие нулевой точке спектра (11) (левая граница – вертикальная прямая). Цифры на границе указывают значения параметров n и j в функциях, определяющих соответствующую границу (см. (17), (18)). Отметим, что собственные функции, отвечающие указанным точкам спектра имеют пространственно-неоднородную форму.

В случае оператора сжатия (3) схема построения областей устойчивости решений начально-краевой задачи (9)-(10) остается прежней, однако в этом случае зависимость b от K и T более сложная.

3. Бифуркация пространственно-неоднородных решений начально-краевой задачи (1)-(2)

Рассмотрим случай растяжения ($\alpha > 1$). При заданных D, γ выберем параметры b_*, T_* таким образом, чтобы они соответствовали точке границы области устойчивости решений начально-краевой задачи (9)-(10) и при этом пучок операторов (11) имел одну пару комплексно сопряженных точек спектра $\pm i\omega_*$, $\omega_* > 0$, которым отвечают собственные функции $u_*(\rho), \bar{u}_*(\rho) \in H^2(K_R)$, $\|u_*(\rho)\|_{L_2} = 1$. Сопряженным с (11) пучком операторов $P^*(\lambda)$: $(P(\lambda)u(\rho,\phi), v(\rho,\phi))_{L_2} =$
$(u(\rho,\phi), P^*(\lambda)v(\rho,\phi))_{L_2}$, $u(\rho,\phi), v(\rho,\phi) \in H^2(K_R)$ будет пучок операторов

$$P^*(\lambda)v(\rho,\phi) \equiv \lambda v(\rho,\phi) + v(\rho,\phi) - D\Delta_{\rho,\phi}v(\rho,\phi) + b(\rho)Q_{\alpha_*}v(\rho,\phi)e^{-\lambda T}, \alpha_* = 1/\alpha. \tag{21}$$

При этом $\pm i\omega_*$ также являются точками спектра, а отвечающие им собственные функции $v_*(\rho), \bar{v}_*(\rho) \in H^2(K_R)$ удовлетворяют условию

$$(P'(i\omega_*)u_*(\rho), v_*(\rho))_{L_2} = 1, \ (P'(i\omega_*)u_*(\rho), \bar{v}_*(\rho))_{L_2} = 0. \tag{22}$$

Отметим, что функции $v_*(\rho)$, $\bar{v}_*(\rho)$ строятся по матрицам $P^{(n)*}(i\omega_*, \alpha)$, сопряженным с (14).

По b_* выберем K_* и $u_* = u(K_*, \gamma)$, удовлетворяющими равенствам (8), (15). Отметим, что такой выбор может быть неоднозначным. Положим теперь $K = K_* + \varepsilon$, где ε – малый параметр, и исследуем возможность бифуркации из состояния равновесия

$$u_*(\varepsilon) = u_*(K_* + \varepsilon, \gamma) = u_* + \varepsilon u_{*1} + ..., u_{*1} = u_*/(1 + b_*)$$

пространственно-неоднородных периодических решений начально-краевой задачи (1)-(2) при изменении параметра ϵ. Для анализа воспользуемся методом инвариантных (центральных) многообразий [7] и теорией нормальных форм обыкновенных дифференциальных уравнений [8].

Отметим, что теперь в (8), (12), (13), (21)

$$b(\rho) = b(\varepsilon) = (K_* + \varepsilon)\gamma \sin(Q_\alpha u_*(\varepsilon)) = b_* + \varepsilon b_1 + ..., \ b_1 = \gamma \sin(u_*) + u_*\gamma \cos(u_*)/(1 + b_*),$$

$$b_2(\rho) = b_2(\varepsilon) = -(K_* + \varepsilon)\gamma \cos(Q_\alpha u_*(\varepsilon)) = -(K_* + \varepsilon)\gamma \cos(u_*(\varepsilon)).$$

Соответственно в дальнейшем будем использовать обозначение пучка операторов $P(\lambda) \equiv P(\lambda; \varepsilon)$. Обозначим $\lambda(\varepsilon) = i\omega_* + \varepsilon\lambda_1 + ...$, точку спектра оператора $P(\lambda; \varepsilon)$, удовлетворяющую условию $\lambda(0) = i\omega_*$, а соответствующую ей собственную функцию $u_*(\rho, \varepsilon) = u_*(\rho) + \varepsilon u_1(\rho) + ...$. $\lambda(\varepsilon)$ и $u_*(\rho, \varepsilon)$ аналитически зависит от ε. Начально-краевая задача (9)-(10) имеет двумерное пространство решений, которое можно записать в следующей форме

$$u(\rho, s, z, \bar{z}; \varepsilon) = u_*(\rho, \varepsilon)e^{\lambda(\varepsilon)s}z + \bar{u}_*(\rho, \varepsilon)e^{\bar{\lambda}(\varepsilon)s}\bar{z}, \ \dot{z} = \lambda(\varepsilon)z,$$

$$z = z(t) \in \mathbb{C}, \ u_*(\rho, \varepsilon)e^{\lambda(\varepsilon)s} \in H_0(K_R; -T_*, 0). \tag{23}$$

Остальные решения начально-краевой задачи (9)-(10) экспоненциально затухают при $t \longrightarrow \infty$.

Начально-краевая задача (6)-(8) в окрестности нулевого состояния равновесия имеет локальное экспоненциально устойчивое инвариантное многообразие (центральное многообразие), поведение решений на котором определяется поведением решений двумерной системы обыкновенных дифференциальных уравнений.

Для построения центрального многообразия и системы дифференциальных уравнений на нем воспользуемся подходом работы [9] и перейдем от (6)-(8) к эквивалентной начально-краевой задаче в области $(0 \le \rho \le R, 0 \le \phi \le 2\pi, -T_* \le s \le 0)$, $t \ge 0$, положив $w(\rho, \phi, s, t) = u(\rho, \phi, t + s)$,

$$w_t = w_s \tag{24}$$

$$w_s(\rho, \phi, 0, t) = -w(\rho, \phi, 0, t) + D\Delta_{\rho\phi}w(\rho, \phi, 0, t) - b(\varepsilon)Q_\alpha w(\rho, \phi, -T_*) +$$
$$+ b_2(\varepsilon)(Q_\alpha w(\rho, \phi, -T_*, t))^2/2 + b(\varepsilon)(Q_\alpha w(\rho, \phi, -T_*, t))^3/6 + ..., \tag{25}$$

$$w_\rho(R, \phi, s, t) = 0, \ w(\rho, \phi, s, t) = w(\rho, \phi + 2\pi, s, t), \ w_\phi(\rho, \phi, s, t) = w_\phi(\rho, \phi + 2\pi, s, t),$$
$$w(\rho, \phi, s, 0) = w_0(\rho, \phi, s) \in H_0(K_R; -T_*, 0). \tag{26}$$

Центральное многообразие в силу специфики рассматриваемого “критического случая” потери устойчивости нулевым решением (6)-(8) не будет зависеть от переменной ϕ. На это указывает также структура решений (23). Будем строить центральное многообразие и дифференциальные уравнения траекторий на нем в виде разложения по

$$W(\rho, s, z, \bar{z}; \varepsilon) = u_*(\rho, \varepsilon)e^{\lambda(\varepsilon)s}z + \bar{u}_*(\rho, \varepsilon)e^{\bar{\lambda}(\varepsilon)s}\bar{z} + w_{20}(.)z^2 + w_{11}(.)z\bar{z} + w_{02}(.)\bar{z}^2 + w_{30}(.)z^3 +$$
$$+ w_{21}(.)z^2\bar{z} + w_{12}(.)\bar{z}^2z + w_{03}(.)\bar{z}^3 + ..., \ w_{jk}(.) = \bar{w}_{jk}(.), w_{jk}(.) = w_{jk}(\rho, s; \varepsilon), \tag{27}$$

$$\dot{z} = \lambda(\varepsilon)z + d_{21}(\varepsilon)z^2\bar{z} + \ldots = Z(z, \bar{z}, \varepsilon). \tag{28}$$

В (27), (28) точками обозначены слагаемые, имеющие по $z, \bar{z}$ более высокий порядок малости. Дифференциальное уравнение для $\bar{z}$ получается простым сопряжением уравнения (28).

Условие принадлежности траекторий уравнения (28) в силу (27) начально-краевой задаче (24)-(26) дает тождество

$$W_t(.) = W_z(.)Z(z, \bar{z}; \epsilon) + W_{\bar{z}}(.)\bar{Z}(z, \bar{z}; \epsilon) \equiv W_s(.), \tag{29}$$

$$W_s(\rho, 0, z, \bar{z}; \epsilon) \equiv -W(\rho, 0, z, \bar{z}; \epsilon) + D\Delta_\rho W(\rho, 0, z, \bar{z}; \epsilon) - b(\epsilon)Q_\alpha W(\rho, -T_*, z, \bar{z}; \epsilon) -$$

$$+ b_2(\epsilon)(Q_\alpha W(\rho, -T_*, z, \bar{z}; \epsilon))^2/2 + b(\epsilon)(Q_\alpha W(\rho, -T_*, z, \bar{z}; \epsilon))^3/6 + \ldots, \tag{30}$$

$$W_\rho(R, s, z, \bar{z}; \epsilon) = 0, \ \Delta_\rho W(\rho) \equiv \frac{1}{\rho}\frac{d}{d\rho}(\rho\frac{dW(\rho)}{d\rho}), \tag{31}$$

которое должно выполняться тождественно при различных степенях $z, \bar{z}, \varepsilon z, \varepsilon\bar{z}$. Приравняем сначала слева и справа в (29)-(31) коэффициенты при εz. В результате получим для определения λ_1 и $u_1(\rho)$ следующую краевую задачу

$$P(i\omega_*; 0)u_1(\rho) \equiv (i\omega_* + 1)u_1(\rho) - D\Delta_\rho u_1(\rho) + b_* e^{-i\omega_* T_*}Q_\alpha u_1(\rho) =$$

$$= -\lambda_1 P'(i\omega_*; 0)u_*(\rho) - b_1 e^{-i\omega_* T_*}Q_\alpha u_*(\rho), \ \frac{du_1(\rho)}{d\rho}|_{\rho=R} = 0 \tag{32}$$

Отметим, что тождество (29) выполняется автоматически. В силу $P(i\omega_*; 0)u_*(\rho) = 0$ и равенств (22) однозначную разрешимость (32) обеспечивают условия

$$\lambda_1 = -(b_1 e^{-i\omega_* T_*}Q_\alpha u_*(\rho), v_*(\rho))_{L_2} = b_1/b_*((1 + i\omega_*)(u_*(\rho), v_*(\rho))_{L_2} + (u_*'(\rho), v_*'(\rho))_{L_2}),$$

$$(P'(i\omega_*; 0)u_1(\rho), v_*(\rho))_{L_2} = 0.$$

Определяя теперь $u_1(\rho)$ в виде ряда по функциям $R_{0j}(\rho)$, получим для определения коэффициентов разложения линейную алгебраическую систему в $\tilde{l}_2$, которая имеет единственное решение, принадлежащее $\tilde{l}_2^2$. Аналогичным образом определяются $\lambda_j, u_j(\rho), j = 2, 3, \ldots$.

Приравняем теперь слева и справа в (29)-(31) коэффициенты при z^2. В результате получим для определения $w_{20}(.)$ краевую задачу вида

$$2\lambda(\varepsilon))w_{20}(\rho, s; \varepsilon)) = w_{20s}(\rho, s; \varepsilon)) \tag{33}$$

$$w_{20s}(\rho, 0; \varepsilon) = -w_{20}(\rho, 0; \varepsilon) + D\Delta_\rho w_{20}(\rho, 0; \varepsilon) - b(\varepsilon)Q_\alpha w_{20}(\rho, -T_*; \varepsilon) +$$

$$+ b_2(\varepsilon)(Q_\alpha u_*(\rho; \varepsilon))^2 e^{-2\lambda(\varepsilon)T_*}/2, \ w_{20\rho}(R, s; \varepsilon) = 0. \tag{34}$$

Решением (33)-(34) будет функция $w_{20}(\rho, s, \varepsilon) = v_{20}(\rho, \varepsilon)e^{2\lambda(\varepsilon)s}$, где $v_{20}(\rho, \varepsilon)$ есть решение краевой задачи

$$P(2\lambda(\varepsilon); \varepsilon)v_{20}(\rho; \varepsilon) = b_2(\varepsilon)(Q_\alpha u_*(\rho; \varepsilon))^2 e^{-2\lambda(\varepsilon)T_*}/2, \ v_{20\rho}(R; \varepsilon) = 0.$$

Решение однозначно определяется в виде

$$v_{20}(\rho, \varepsilon) = \sum_{j=0}^{\infty} v_j(\varepsilon)R_{0j}(\rho), \ v_{20}(\varepsilon) = (v_0(\varepsilon), v_1(\varepsilon), \ldots) \in \tilde{l}_2^2,$$

$v_{20}(\varepsilon)$ решение линейного уравнения в $\tilde{l}_2$

$$P^{(0)}(2\lambda(\varepsilon), \alpha; \varepsilon)v_{20}(\varepsilon) = f_{20}(\varepsilon), \ f_{20}(\varepsilon) = (f_0(\varepsilon), f_1(\varepsilon), \ldots),$$

$$f_j(\varepsilon)) = b_2(\varepsilon)e^{-2\lambda(\varepsilon))T_*}((Q_\alpha u_*(\rho;\varepsilon))^2, R_{0j}(\rho))_{L_2}/2,$$

с матрицей, определенной в (13). Аналогичным образом определяется функция $w_{11}(\rho, s, \varepsilon)) = v_{11}(\rho, \varepsilon))e^{(\lambda(\varepsilon))+\bar{\lambda}(\varepsilon)))s}$, где

$$v_{11}(\rho, \varepsilon)) = \sum_{j=0}^{\infty} v_j(\varepsilon)R_{0j}(\rho),\ \ v_{11}(\varepsilon) = (v_0(\varepsilon), v_1(\varepsilon), \dots) \in \tilde{l}_2^2,$$

$v_{11}(\varepsilon)$ решение линейного уравнения в $\tilde{l}_2$

$$P^{(0)}(\lambda(\varepsilon) + \bar{\lambda}(\varepsilon), \alpha; \varepsilon)v_{11}(\varepsilon) = f_{11}(\varepsilon),\ \ f_{11}(\varepsilon) = (f_0(\varepsilon), f_1(\varepsilon), \dots),$$

$$f_j(\varepsilon)) = b_2(\varepsilon)e^{(\lambda(\varepsilon)+\bar{\lambda}(\varepsilon))T_*}(Q_\alpha u_*(\rho;\varepsilon)Q_\alpha \bar{u}_*(\rho;\varepsilon), R_{0j}(\rho))_{L_2}.$$

Рассмотрим определение функции $w_{21}(\rho, s; \epsilon)$. Приравняв слева и справа в (29)-(31) коэффициенты при $z^2\bar{z}$, получим краевую задачу

$$u_*(\rho, \varepsilon)e^{\lambda(\varepsilon)s}d_{21}(\varepsilon) + (2\lambda(\varepsilon) + \bar{\lambda}(\varepsilon))w_{21}(\rho, s; \varepsilon) = w_{21s}(\rho, s; \varepsilon), \tag{35}$$

$$w_{21s}(\rho, 0, \varepsilon) = -w_{21}(\rho, 0, \varepsilon) + D\Delta_{\rho\phi}w_{21}(\rho, 0; \varepsilon) - b(\varepsilon)Q_\alpha w_{21}(\rho, -T_*; \varepsilon)+$$
$$+(b_2(\varepsilon)(Q_\alpha v_{20}(\rho;\varepsilon)Q_\alpha \bar{u}_*(\rho, \varepsilon) + Q_\alpha v_{11}(\rho;\varepsilon)Q_\alpha u_*(\rho, \varepsilon)+$$
$$+ b(\varepsilon)(Q_\alpha u_*(\rho, \varepsilon)^2 Q_\alpha \bar{u}_*(\rho, \varepsilon)/2)e^{-(2\lambda(\varepsilon)+\bar{\lambda}(\varepsilon))T_*},\ \ w_{21\rho}(R, s; \varepsilon) = 0. \tag{36}$$

Краевая задача (35)-(36) при $\varepsilon = 0$ не разрешима. Разрешимости добиваемся выбором $d_{21}(\varepsilon)$, который при этом определяется однозначно. Непрерывное по ε решение (35)-(36) также определяется однозначно в виде разложения по $R_{0j}(\rho)$. При этом коэффициенты разложения однозначно определяются по аналогии с краевой задачей (33)-(34). В дальнейшем используется лишь коэффициент $d_{21}(0)$, поэтому приведем для него выражение

$$d_{21}(0) = (b_2(0)(Q_\alpha v_{20}(\rho; 0)Q_\alpha \bar{u}_*(\rho) + Q_\alpha v_{11}(\rho; 0)Q_\alpha u_*(\rho))+$$
$$+b_*(Q_\alpha u_*(\rho))^2 Q_\alpha \bar{u}_*(\rho)/2, v_*(\rho))_{L_2} e^{-i\omega_* T_*},$$

полученное из условия разрешимости (35)-(36) при $\varepsilon = 0$ с учетом равенств (22).

Величины λ_1, $d_{21}(0)$ были вычислены в точках границы области устойчивости (на рисунке 1 это граница (0 0)). При этом оказалось, что во всех точках $Re\ \lambda_1 > 0$, $Re\ d_{21}(0) < 0$. В связи с этим при $\varepsilon > 0$ уравнение (28) имеет асимптотически устойчивое периодическое решение

$$z(t; \varepsilon^{1/2}) = \varepsilon^{1/2}\rho_* e^{i\tau} + O(\varepsilon), \rho_* = (-Re\lambda_1/Red_{21}(0))^{1/2},$$

$$\dot{\tau} = \omega_*(\varepsilon) = \omega_* + \varepsilon(Im\lambda_1 + Imd_{21}(0)\rho_*^2) + O(\varepsilon^2),$$

в котором $z(t; \varepsilon^{1/2})$ и $\omega_*(\varepsilon)$ аналитические функции $\varepsilon^{1/2}$ и ε соответственно.

Этому решению в начально-краевой задаче (6)-(8) соответствует согласно (27)-(28) асимптотической устойчивости периодическое решение вида

$$u(\rho, \omega_*(\varepsilon)(t + s); \varepsilon^{1/2}) = \varepsilon^{1/2}(u_*(\rho)e^{i\omega_*(\varepsilon)(t+s)} + \bar{u}_*(\rho)e^{-i\omega_*(\varepsilon)(t+s)}) + O(\varepsilon),$$

которое представляет собой стоячую волну периода $2\pi/\omega_*(\varepsilon)$. Указанное решение является аналитической функцией $\varepsilon^{1/2}$.

References

[1] S. A. Akhmanov, M. A. Vorontsov, and V. J. Ivanov, "Krupnomasshtabnye poperechnye nelineynye vzaimodeistviya v lazernyh puchkah; novye tipy nelineinyh voln, vozniknovenie opticheskoi turbylentnosti", *Pisma v ZHETPH*, vol. 47, no. 12, pp. 611–614, 1988, In Russian.

[2] S. A. Akhmanov and M. A. Vorontsov, "Neustojchivosti i struktury v kogerentnyh nelinejno-opticheskih sistemah, ohvachennyh dvumernoj obratnoj svyaz'yu", *Scince*, vol. 9, pp. 228–238, 1989, In Russian.

[3] S. A. Akhmanov, M. A. Vorontsov, V. Y. Ivanov, A. V. Larichev, and N. I. Zheleznykh, "Controlling transverse wave interactins in nonlinear optics: generation and interaction of spatiotemporal structures", vol. 9, no. 1, pp. 78–90, 1992.

[4] A. V. Razgulin and T. E. Romanenko, "Vrashchayushchiesya volny v parabolicheskom funkcional'no-differencial'nom uravnenii s povorotom prostranstvennogo argumenta i zapazdyvaniem", *ZH. vychisl. matem. i matem. fiz.*, vol. 53, no. 11, pp. 1804–1821, 2013, In Russian.

[5] E. P. Kubyshkin and V. A. Kulikov, "Analysis of the conditions for the emergence of spatially inhomogeneous structures of light waves in optical information transmission systems", *Modeling and Analysis of Information Systems*, vol. 26, no. 2, pp. 297–305, 2019, In Russian.

[6] Y. I. Neymark, "D-razbienie prostranstva kvazipolinomov (k ustojchivosti linearizovannyh raspredelennyh sistem)", *PMM*, vol. 13, no. 4, pp. 349–380, 1949.

[7] J. Marsden and M. McCracken, *The Bifurcation of the Birth of a Cycle and Its Applications*, 1980.

[8] A. D. Bryuno, "Lokal'nyj metod nelinejnogo analiza differencial'nyh uravnenij", *Scince*, 1979.

[9] E. P. Kubyshkin and A. R. Moriakova, "Features of Bifurcations of Periodic Solutions of the Ikeda Equation", *Russian Journal of Nonlinear Dynamics*, vol. 14, no. 3, pp. 301–324, 2018.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 27, NO. 2, 2020
journal homepage: www.mais-journal.ru

THEORY OF COMPUTING

Dynamically Changing Parallelism with the Asynchronous Sequential Data Flows

A. I. Legalov[1], I. V. Matkovskii[1], M. S. Ushakova[1], D. S. Romanova[1]

DOI: 10.18255/1818-1015-2020-2-164-179

[1]Siberian Federal University, 79 Svobodny pr., 660041 Krasnoyarsk, Russia.

MSC2020: 68N15, 68Q10, 68Q85
Research article
Full text in Russian

Received May 27, 2020
After revision June 8, 2020
Accepted June 10, 2020

A statically typed version of the data driven functional parallel computing model is proposed. It enables a representation of dynamically changing parallelism by means of asynchronous serial data flows. We consider the features of the syntax and semantics of the statically typed data driven functional parallel programming language Smile that supports asynchronous sequential flows. Our main idea is to apply the Hoar concept of communicating sequential processes to the computation control on the data readiness. It is assumed that on the data readiness a control signal is emitted to inform the processes about the occurrence of certain events. The special feature of our approach is that the model is extended with the special asynchronous containers that can generate events on their partial filling. These containers are a stream and a swarm, each of which has its own specifics. A stream is used to process data which have identical type. The data comes sequentially and asynchronously at arbitrary time moments. The number of the incoming data elements is initially unknown, so the processing completes on the signal of the end of the stream. A swarm is used to contain independent data of the same type and may be used for the massive parallel operations performing. Unlike a stream, the swarm's size is fixed and known in advance. General principles of the operations with the asynchronous sequential flows with an arbitrary order of data arrival are described. The use of the streams and the swarms in various situations is considered. We propose the language constructions which allow us to operate the swarms and streams and describe the specifics of their application. We provide the sample functions to illustrate the use of the different approaches to description of the parallelism: recursive processing of the asynchronous flows, processing of the flows in an arbitrary or predefined order of operations, direct access and access by the reference to the elements of the streams and swarms, pipelining of calculations. We give a preliminary parallelism assessment which depends on the ratio of the rates of data arrival and their processing. The proposed methods can be used in the development of the future languages and tool-kits of architecture-independent parallel programming.

Keywords: parallel computations, asynchronous computations, static typing, dynamically changing parallelism.

INFORMATION ABOUT THE AUTHORS

Alexander I. Legalov correspondence author	orcid.org/0000-0002-5487-0699. E-mail: legalov@mail.ru Head of the scientific and educational laboratory of programming technologies, professor of the department of computer engineering, doctor of technical sciences, professor.
Ivan V. Matkovskii correspondence author	orcid.org/0000-0002-4801-7982. E-mail: alpha900i@mail.ru Senior teacher.
Mariya S. Ushakova correspondence author	orcid.org/0000-0003-4234-2714. E-mail: ksv@akadem.ru Assistent of the department of computer engineering.
Darya S. Romanova correspondence author	orcid.org/0000-0002-9020-4802. E-mail: daryaooo@mail.ru Graduate student.

For citation: A. I. Legalov, I. V. Matkovskii, 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.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 27, № 2, 2020
сайт журнала: www.mais-journal.ru

THEORY OF COMPUTING

Динамически изменяющийся параллелизм с асинхронно-последовательными потоками данных

А. И. Легалов[1], И. В. Матковский[1], М. С. Ушакова[1], Д. С. Романова[1]

DOI: 10.18255/1818-1015-2020-2-164-179

[1]Сибирский федеральный университет, 660041, Красноярский край, г. Красноярск, пр. Свободный, 79.

УДК 004.042
Научная статья
Полный текст на русском языке

Получена 27 мая 2020 г.
После доработки 8 июня 2020 г.
Принята к публикации 10 июня 2020 г.

Предлагается статически типизированная версия модели функционально-потоковых параллельных вычислений, которая за счет использования асинхронных последовательных потоков обеспечивает представление динамически изменяющегося параллелизма. Рассмотрены особенности синтаксиса и семантики статически типизированного языка функционально-потокового параллельного программирования Smile, обеспечивающие поддержку асинхронных последовательных потоков. Основная идея подхода базируется на использовании концепции взаимодействующих последовательных процессов Т. Хоара применительно к управлению по готовности данных. Предполагается, что готовность данных сопровождается выдачей управляющих сигналов, информирующих процессы о свершении тех или иных событий. Отличительной особенностью подхода является включение в модель специальных асинхронных контейнеров, которые могут порождать события по частичному заполнению. Этими контейнерами являются поток и рой, каждый из которых имеет свою специфику. Поток используется для обработки данных одного типа, поступающих последовательно и асинхронно в произвольные промежутки времени. Размерность поступающих данных изначально неизвестна, поэтому завершение обработки осуществляется по признаку конца потока. Рой используется для описания независимых данных одного типа, над которыми возможно выполнение массовых параллельных операций. В отличие от потока, его размерность фиксирована и известна заранее. В работе описаны общие принципы организации асинхронных последовательных потоков с произвольным поступлением данных. Рассматривается использование потоков и роев в различных ситуациях. Предлагаются языковые конструкции, позволяющие описывать работу с роями и потоками и особенности их применения. Представлены примеры функций, при реализации которых использованы различные подходы к описанию параллелизма: рекурсивная обработка асинхронных потоков, обработка потоков при недетерминированном и упорядоченном выполнении операций, прямое и ссылочное обращение к элементам потоков и роев, конвейеризация вычислений. Дается предварительная оценка параллелизма в зависимости от временных соотношений между темпом поступления данных и скоростью их обработки. Предложенные методы могут быть использованы при разработке перспективных языковых и инструментальных средств архитектурно-независимого параллельного программирования.

Ключевые слова: параллельные вычисления, асинхронные вычисления, статическая типизация, динамически-изменяющийся параллелизм.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Александр Иванович Легалов автор для корреспонденции	orcid.org/0000-0002-5487-0699. E-mail: legalov@mail.ru Руководитель научно-учебной лаборатории технологий программирования, профессор кафедры вычислительной техники, доктор технических наук, профессор.
Иван Васильевич Матковский автор для корреспонденции	orcid.org/0000-0002-4801-7982. E-mail: alpha900i@mail.ru Старший преподаватель.
Мария Сергеевна Ушакова автор для корреспонденции	orcid.org/0000-0003-4234-2714. E-mail: ksv@akadem.ru Ассистент кафедры вычислительной техники.
Дарья Сергеевна Романова автор для корреспонденции	orcid.org/0000-0002-9020-4802. E-mail: daryaooo@mail.ru Аспирант.

Для цитирования: A. I. Legalov, I. V. Matkovskii, 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.

Введение

Разработка архитектурно-независимых параллельных программ определяется двумя следующими подходами:

- разработкой и отладкой последовательных программ, не связанных с параллельными вычислительными системами (ПВС), с последующим распараллеливанием под целевую архитектуру;
- изначальной разработкой программ или алгоритмов, описывающих максимальный параллелизм решаемой задачи с последующим «сжатием» этого параллелизма к целевой архитектуре.

В реальной ситуации чаще используется промежуточное решение, когда параллельная программа сразу же разрабатывается с учетом особенностей целевой архитектуры. Однако, в отличие от первых двух вариантов, формируемый код оказывается жестко связан с целевым решением, что затрудняет его перенос на другие ПВС.

Вместе с тем, независимо от используемых подходов следует отметить, что практически в любом из этих трех вариантов параллелизм программы фиксируется за счет использования не только изначально заданного базиса операций, но и из-за методов описания параллельных процессов. Зачастую это не связано с ориентацией на архитектурную зависимость, а определяется спецификой как моделей вычислений, так и построенных на их основе языковых и инструментальных средств.

Фиксация параллелизма как сверху (максимальным параллелизмом), так и снизу (последовательным выполнением) ведет к семантическому разрыву на уровне реального вычислителя. То есть, при выполнении программы в реальных вычислительных ресурсах происходит потеря эффективности и сбалансированности из-за того, что зафиксированные при разработке характеристики параллельного алгоритма вступают в противоречие с динамическими характеристиками подсистем ПВС. Именно поэтому максимальный параллелизм зачастую ужимается при доводке программы под особенности вычислителя, что ведет к решению задачи, противоположной распараллеливанию последовательных программ. Это, в свою очередь, ведет к потере эффективности процесса разработки параллельного программного обеспечения, не позволяя писать программу один раз и для различных параллельных архитектур.

В связи с этим актуальной является задача поиска моделей параллельных вычислений и построение на их основе языковых и инструментальных средств, обеспечивающих адаптивную подстройку параллелизма однажды написанной программы для ее эффективного наложения на различные вычислительные ресурсы.

Одним из таких решений является использование данных по мере их поступления. То есть, без накопления в массивах перед последующими вычислениями. Предполагается, что их обработка происходит по мере готовности отдельных элементов. Параллелизм при обработке таких данных зависит как от темпа поступления этих элементов в массивы, так и от времени выполнения их последующей обработки. Отношение между этими временами позволяет рассматривать разные уровни параллелизма соответствующих алгоритмов — от последовательных вычислений до неограниченного распараллеливания. Одна из таких структур данных, названная асинхронным списком, была предложена в ходе расширения функционально-потоковой модели параллельных вычислений (ФПМПВ) [1].

Вместе с тем следует отметить, что данный вид асинхронных потоков данных обладает следующим недостатком: порядок следования аргументов, поступающих на выполнение, может не совпадать с порядком их следования в выходном потоке, используемом для сбора результатов. Это изменение может быть некритичным для некоторых алгоритмов. Однако в большинстве случаев подобная ситуация недопустима. Одним из вариантов решения данной проблемы является организация упорядоченных очередей, когда результаты формируются в той же последовательности,

что и порождающие их аргументы. Однако в этой ситуации происходит неявная синхронизация потока асинхронно поступающих данных, что, в свою очередь, ведет к замедлению их обработки.

Для решения проблемы предлагается подход, базирующийся на расширении возможностей параллельного списка ФПМПВ. Предполагается, что данные, попадающие в этот список, могут сразу же поступать на обработку, после которой они устанавливаются на те же позиции в списке результатов. Дополнительное изменение семантики можно внести также и в рассмотренный ранее асинхронный список. Применение предлагаемых изменений позволяет создавать новую разновидность модели вычислений, описывающей асинхронные алгоритмы с динамически изменяемым параллелизмом.

В работе рассматриваются особенности семантики операторов, обеспечивающих поддержку асинхронных потоковых вычислений, а также их отображение на синтаксис и семантику статически типизированного языка функционально-потокового параллельного программирования Smile. Приводятся примеры, демонстрирующие специфику предлагаемых конструкций. Дается предварительная оценка параллелизма в зависимости от временных соотношений между темпом поступления данных и скоростью их обработки.

1. Основные идеи, определяющие подход

Основная идея подхода базируется на концепции, предложенной Хоаром [2], в которой параллелизм описывается как взаимодействие процессов, порождаемых через последовательно формируемые события. Их последовательность обуславливается тем, что моменты возникновения событий считаются мгновенными и на оси времени выстраиваются в очередь. Это позволяет игнорировать одновременность. Считается, что вместо одновременности событий появляется их недетерминированность, когда два конкурирующих события могут появиться в произвольной последовательности. Аналогичный подход также используется в ряде систем моделирования на основе сетей Петри [3].

В управлении по готовности данных порождение таких событий можно связать с моментами порождения данных. Отделение событий от данных можно рассматривать как поток управляющих сигналов, взаимодействие которых позволяет формировать различные стратегии управления вычислениями [4, 5]. На основе данного подхода в рамках концепции функционально-потокового параллельного программирования предложена модель событийного процессора [6], осуществляющего обход информационного графа функционально-потоковой параллельной программы и выполнение обработки данных на основе реагирования только на управляющие сигналы.

Использование последовательности асинхронных сигналов позволяет описать параллелизм без явных распараллеливающих схем. Предложенные в ФПМПВ асинхронные списки [1] позволяют описывать вычисления через последовательные запуски рекурсивных вызовов. Показано также, что использование асинхронных списков в этом случае обеспечивает динамическое изменение параллелизма в зависимости от отношения между темпом поступления данных и временем выполнения операций обработки данных от максимального параллелизма до последовательных вычислений. То есть, ориентация на асинхронно поступающие данные и последовательности сигналов, информирующих об их поступлении, позволяет более гибко описывать параллельные алгоритмы и в дальнейшем адаптировать их к конкретным условиям, используя только один способ описания алгоритма на основе последовательно обрабатываемых асинхронных потоков. Недостаток асинхронных списков, связанный с недетерминированностью поступления в них обрабатываемых данных предлагается исправить за счет модификации параллельных списков, расширенных функциями, поддерживающими дополнительные механизмы управления на основе сигналов о готовности данных.

Использование событийного управления вычислениями в системах с управлением на основе готовности данных было предложено в [6]. В рамках этой концепции разработан интерпретатор функционально-потоковых параллельных программ, написанных на языке программирования Пифагор [7]. Применение событийного управления позволило отделить граф управления вычислениями от информационного графа программы. При этом стало возможным изменение стратегии управления вычислениями за счет изменения управляющего графа без изменения информационного графа программы.

Дальнейшее развитие представленных работ связано с расширением семантики ФПМПВ и введением в модель и язык статической типизации данных, что обеспечивает более гибкую трансформацию в другие параллельные архитектуры. Внесенные изменения привели к созданию статически типизированной модели функционально-потоковых параллельных вычислений (СТМФППВ) и разработке на основе этой модели статически типизированного языка функционально-потокового параллельного программирования Smile [8].

2. Описание ключевых понятий модели вычислений, обеспечивающих поддержку асинхронных последовательных потоков

Предлагаемые новые понятия по сути расширяют возможности ряда программоформирующих операторов ФПМПВ. Однако использование в новой модели и языке программирования статической типизации данных вместо динамической типизации, с одной стороны, накладывает свои ограничения, но, с другой стороны, предоставляет дополнительные возможности по трансформации функционально-потоковых параллельных программ в программы для реальных архитектур.

В СТМФППВ, по сравнению с ФПМПВ, изменилась семантика контейнерных типов данных, обеспечивающих поддержку массовых операций. В частности, для поддержки анализа типов во время компиляции, вместо списка данных появился вектор (vector), все элементы которого должны быть одного предопределенного (именованного) типа. Однотипные элементы также должны быть у роя и потока. Это позволяет ввести над этими конструкциями массовые поэлементные операции, а также, наряду с ними, независимо от контейнера, выполнять функции, воспринимающие контейнерные типы как единое целое.

Использование статической типизации также привело к разделению оператора интерпретации на два разных вида: одиночный (одноаргументный) и групповой (массовый, поэлементный). Одиночный оператор интерпретации, обозначаемый в текстовом представлении, как и ранее [9], через «:» (постфиксная форма) или «^» (префиксная форма) предназначен для задания обычных функций, воспринимающих аргумент в качестве единого целого. Массовый оператор интерпретации используется для задания вычислений над каждым однотипным элементом контейнера, порождая на выходе контейнер с элементами тип которых соответствует типу результата выполняемой функции. Обозначается двойным значком «::» для постфиксной или «^^» для префиксной форм соответственно.

Использование разных обозначений позволяет однозначно применять функцию с одним и тем же именем в разных контекстах. Например, функция вычитания «-» над аргументом (10, -3), воспринимаемом как вектор, состоящий из двух целых чисел, порождает следующие значения:

```
// двуместная функция вычитания над одним аргументом
(10, -3):- => 13

// функция смены знака, массово применяемая
// к двум однотипным аргументам
(10, -3)::- => (-10, 3)
```

Разделение оператора интерпретации на массовый и одноаргументный позволяет ввести более гибкий одноаргументный набор дополнительных функций для потока и роя, обеспечивающих обработку асинхронно поступающих данных.

2.1. Организация асинхронных последовательных потоков с произвольным поступлением данных

Введенное в СТМФППВ понятие потока (stream), расширяет концепцию ранее предложенного асинхронного списка. Основная идея, связанная с асинхронным поступлением данных, сохраняется. Однако предполагается, что все элементы имеют один и тот же именованный тип, который, в свою очередь, не может являться потоком или роем. Это вполне соответствует концепциям универсальных статически типизированных языков. Поток можно рассматривать как сущность (рисунок 1), к основным характеристикам которой относятся:

- при появлении в потоке хотя бы одного готового элемента данных, он порождает сигнал, информирующий об его готовности;
- готовый элемент может быть прочитан из потока для обработки;
- если во время обработки элемента, выбранного из потока в него поступают новые элементы данных, они также могут асинхронно выбираться из потока в порядке поступления и обрабатываться параллельно;
- параллельно обрабатываемые элементы потока могут поступать после обработки в другой поток, тип которого определяется типом результата функции, при этом порядок их поступления может отличаться от первоначального в зависимости от времени обработки;
- поток можно проверить на отсутствие дальнейшего поступления данных, что позволяет завершить работу с ним.

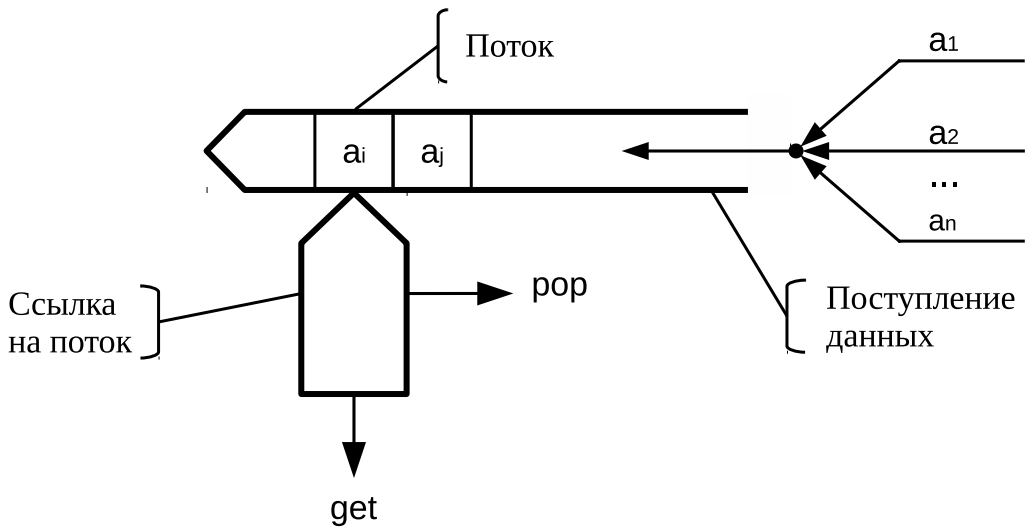


Fig. 1. General scheme of the stream

Рис. 1. Обобщенная схема потока

2.1.1. Основные операции с потоком

Описание потока в языке программирования Smile определяется следующим синтаксисом:

```
поток = имя_типа_элементов "{}".
```

Выполнение операции с потоками может осуществляться с использованием массовой операции интерпретации. В этом случае все элементы обрабатываемого потока после выполнения над ними функции поступают в формируемый на выходе поток с результатами вычислений. Например, вычисление функции `sin` над всеми элементами входного потока `X` с формированием на выходе потока `Y` на языке Smile можно записать следующим образом:

```
X::sin >> Y
```

где поток `X` предварительно описан следующим образом: `X@float{}`. Символ `@` отделяет имя используемой сущности от ее типа. Автоматически формируемый на выходе поток имеет такой же тип, что и тип результата функции `sin` и определяется ее сигнатурой:

```
sin << func float -> float
```

При этом порядок результатов в потоке, обозначенном через `Y` может отличаться от порядка поступления аргументов в поток `X`.

Принято допущение, что операции над потоком не выполняются непосредственно. Это связано с тем, что прямое обращение к потоку может привести к побочным эффектам, изменяющим его состояние, что не позволит корректно взаимодействовать с потоком другим операциям, выполняемым параллельно. Вместо этого одноаргументный оператор интерпретации обращается к потоку через ссылку. Наличие нескольких ссылок на один поток, играющих по сути роль итераторов, позволяет обрабатывать его различным образом в разных частях программы. Синтаксис ссылки на поток в Smile имеет следующий вид:

```
ссылка_на_поток = имя_типа_элементов "{*}".
```

Потоки через ссылки могут передаваться в функции в качестве параметров. Функция вычисления синуса для всех элементов потока в Smile будет выглядеть следующим образом:

```
sinStream << funcdef X@float{*} -> float{*} {
    X::sin:return
}
```

То есть, передача через ссылку допускает массовое выполнение функций. Следует также отметить, что при использовании потока в качестве аргумента массового оператора интерпретации внутри последнего автоматически создается скрытая локальная ссылка на этот поток, что позволяет избежать побочных эффектов.

В большинстве случаев использования массовых операций над потоком недостаточно для гибкого асинхронного программирования, когда требуется поэлементная обработка или объединение данных, поступающих из нескольких потоков. Для этого необходимо применять дополнительные конструкции, во многом аналогичные итераторам. Эти конструкции задаются специальными одноаргументными функциями над потоком. В частности, перед непосредственным доступом к элементу потока необходимо предварительно проверить, что поток еще порождает элементы (по аналогии с проверкой признака конца файла). Это обуславливается тем, что количество элементов, которые порождает поток, может быть заранее неизвестно. Проверка потока на то, что данные в нем

еще порождаются осуществляется функцией is, имеющей следующую сигнатуру:

```
is << func any{*} -> bool
```

где any — ключевое слово, обозначающее любой тип. Функция возвращает значение true, если поток еще может формировать данные или уже содержит их. В противном случае возвращается false.

Для получения данных из потока используется функция get, которая читает элемент потока, стоящий в его очереди первым. Если такой элемент еще не сформировался, функция get ожидает его поступления. При наличии в очереди потока нескольких элементов выбирается только один. При попытке выполнить эту функцию для уже завершенного потока формируется ошибка, ведущая к прерыванию функции. Функция имеет следующую сигнатуру:

```
get << func any{*} -> any
```

Перед тем как прочитать следующий элемент из входного потока необходимо убрать из ссылки уже прочитанный элемент. Для этого используется функция pop. При попытке выполнить эту функцию для уже завершенного потока формируется ошибка, ведущая к прерыванию функции. Функция имеет следующую сигнатуру:

```
pop << func any{*} -> any{*}
```

То есть, она возвращает новую ссылку на тот же поток, но уже без обработанного элемента (этот элемент уже недоступен через возвращаемую ссылку).

В качестве примера использования одноаргументных функций можно рассмотреть нахождение суммы элементов поступающих во входной поток:

```
sum << func X@float{*} -> float {
    if << X:is;
    if^({(X:get,X:pop:sum):+}, 0):return
}
```

Проверка X:is порождает булевское значение true/false, которое используется оператором интерпретации в качестве селектора. При истинном значении выбирается первый элемент кортежа, запускающий левую рекурсию для функции sum. Значение false формируется, когда поток завершен. В этом случае возвращается значение 0. При обратном ходе рекурсивного процесса осуществляется суммирование элементов.

2.1.2. Занесение информации в поток из различных источников

В представленном выше примере суммирования элементов потока, по сути, реализована последовательная рекурсия, так как при обратном ходе накопление суммы осуществляется путем сложения очередного элемента потока с накопленным промежуточным значением. В работе [1] показано, что асинхронный список через последовательные рекурсивные вызовы позволяет реализовать суммирование параллелизм которого, в зависимости от временных соотношений между интенсивностью поступления данных и скоростью их обработки, может достигать максимального, эквивалентного каскадной свертке. Использование потоков в языке программирования Smile позволяет написать аналогичную функцию. При этом наличие возможности создавать хранилища обеспечивает занесение в поток не только исходных данных, но и промежуточных результатов.

Соответствующая функция суммирования значений, поступающих в поток, выглядит следующим образом:

```
// Асинхронное суммирование элементов потока
// с внесением в него результатов промежуточных вычислений
sum << func X@float{*} -> float {
    // Проверка, что данные в поток еще могут поступить
    notEmpty << X:is;
    notEmpty^(
        // Есть хотя бы один элемент
        {block{
            // Элемент выбирается из потока
            a << X:get;
            // Формируется ссылка на следующую позицию
            Y << X:pop;
            // и делается проверка на наличие следующего элемента
            notEmptySecond << Y:is;
            notEmptySecond^(
                {block{
                    // При наличии второго элемента можно его сложить с первым
                    // И через любую доступную ссылку переслать в поток
                    (a, Y:get):+ -> Y;
                    // Также создать новую ссылку и без второго элемента
                    // рекурсивно продолжив вычисления
                    Y:pop:sum:break
                },
                // В противном случае в потоке только один элемент,
                // значение которого и является суммой
                a
            ):break
        }},
        // При отсутствии данных возвращается 0
        0.0
    ):return
}
```

В данной ситуации функция, при наличии хотя бы двух элементов в потоке, суммирует их. Полученная сумма через ссылку пересылается в этот же поток. Процесс рекурсивно повторяется для вновь поступающих элементов, чередующихся с промежуточными вычислениями сумм до момента, когда в потоке останется только одна величина, которая и является окончательной суммой.

2.1.3. Недетерминированность поведения потока при выполнении асинхронных вычислений

Использование потоков позволяет организовывать асинхронные вычисления с динамически изменяемым параллелизмом, зависящим от временных соотношений между интервалами поступления данных в поток и скоростью их обработки функциями, взаимодействующими с потоком. Однако высокая вероятность того, что порядок поступления аргументов не будет совпадать с порядком получения результатов на выходе, не позволяет во многих случаях организовать детерминированные и предсказуемые вычисления. В качестве такого примера можно рассмотреть вычисление

массива данных, поступающих из входного потока, а после обработки направляющихся в выходной поток. Пусть для вычислений используется формула:

```
y[i] = sin(x[i])*sin(x[i])+cos(x[i])*cos(x[i])
```

При использовании потоков в качестве промежуточных хранилищ результатов без особых сложностей организуются конвейерные вычисления (при соответствующих временных соотношениях). Однако в связи с возможностью различного времени выполнения функций над элементами потоков, корректные последовательности значений в результирующем потоке могут быть не получены.

Эта ситуация может быть описана следующим кодом на языке Smile:

```
SumSin2Cos2Stream << func X@float{*} -> float{*} {
    result@float{};
    (X, result):GetStreamResult >> ok;
    result:ok:return
}
```

где:

```
GetStreamResult << func (arg@float{*}, result@float{*})->signal {
    // Проверка потока на возможное поступление данных
    if << arg:is;
    if^(
        // Занесение результата в выходной поток
        // после добавления в него данных
        {block {
           x << arg:get;    // Получение элемента из потока
           s << x:sin; Sin2 << (s,s):*; // Синус в квадрате
           c << x:cos; Cos2 << (c,c):*; // Косинус в квадрате
           // Вычисление текущего значения с передачей в выходной поток
           (Sin2,Cos2):+ -> result;
           // Убирается обработанный элемент из потока
           // и переход к обработке следующего элемента
           (arg:pop, result):GetStreamResult:break}
        },
        // Сигнал без заполнения результата,
        // если данные в поток больше не поступают
        !
    ):return
}
```

Основная функция `SumSin2Cos2Stream` получает данные из входного потока через ссылку `X`. Результат вычислений через ссылку на поток возвращается из функции. Сам поток реализован внутри функции через хранилище `result`, а накопление в нем результатов вычислений происходит в функции `GetStreamResult`, в которую он передается в качестве параметра.

Функция `GetStreamResult` производит основные вычисления для первого текущего элемента, поступившего во входной поток. Полученное значение суммы передается, используя обращение к элементу выходного потока (обозначено через ->). Одновременно с этим происходит рекурсивный вызов функции `GetStreamResult`, в которую ссылка на входной поток передается уже без учета

первого аргумента. Блок `block` используется для локализации группы операторов, из которой только один возвращает результат посредством выполнения функции `break`. Данные в блок поступают через имена, описанные вне его.

При передаче результатов вычислений в новый поток порядок поступления элементов может изменяться относительно первоначального потока, что ведет к появлению недетерминированности вычислений и некорректному результату. Пример показывает, что необходимо расширить модель вычислений конструкциями, обеспечивающими сохранение порядка следования данных и при этом поддерживающими асинхронные взаимодействия.

2.2. Организация упорядоченных данных с сохранением порядка следования

Для сохранения порядка следования при обработке данных необходимо использовать контейнерные типы, обеспечивающие асинхронное формирование отдельных элементов. В ФПМПВ такой сущностью является параллельный список. Однако он поддерживает выполнение только массовых операций над его элементами и не допускает обработки самого списка как единого аргумента. В СТМФППВ вводятся расширения, обеспечивающие поддержку необходимой функциональности. Вместо параллельного списка используется рой, который, наряду с массовыми операциями, как и поток, допускает свое использование в качестве единственного аргумента (рисунок 2).

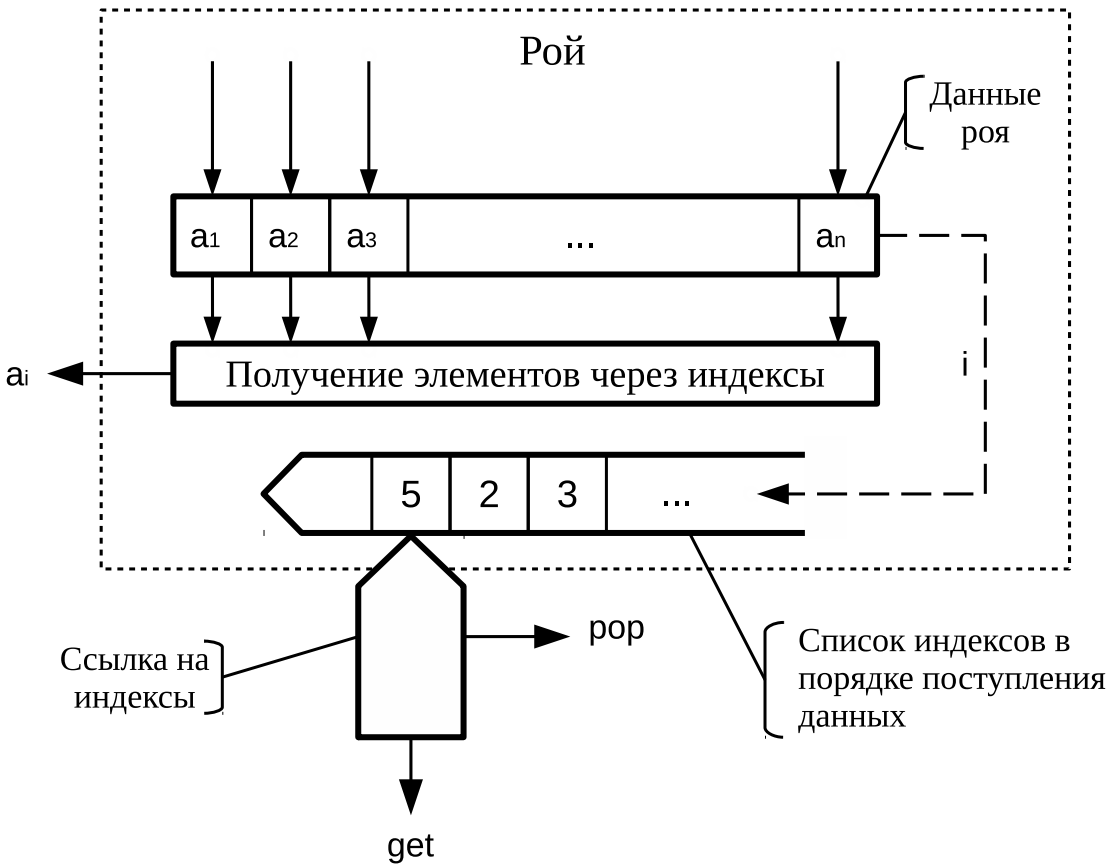


Fig. 2. General scheme of the swarm and its reference

Рис. 2. Обобщенная схема роя и ссылки на него

Спецификой предлагаемой СТМФППВ и разрабатываемой на ее основе статически типизированного языка функционально-потокового параллельного программирования Smile является

наличие информации о типах во время компиляции. Это ведет к изменению алгебры эквивалентных преобразований и семантики многих базовых операций, ориентированных не на интерпретацию исходной программы, а на генерацию кода для целевых архитектур. В частности, запрещается непосредственная вложенность роев, что облегчает анализ аргументов оператора интерпретации во время компиляции и позволяет определить, является ли функция массовой над всеми элементами роя или это функция над всем роем. Ряд преобразований роя для использования в массовых операциях может происходить уже во время компиляции. Для представления роя используется список из элементов, заключенных в квадратные скобки.

Передача роя в функции и возврат их осуществляется, как и для потоков, через ссылки. Синтаксис ссылки на рой в Smile имеет следующий вид:

```
ссылка_на_рой = имя_типа_элементов "[*]".
```

Используя их можно написать следующий вариант функции одновременного упорядоченного вычисления синуса для всех элементов роя:

```
sinSwarm << funcdef X@float[*]->float[*] {
    X::sin:return
}
```

В этой ситуации непосредственное использование роевых функций позволяет избавиться от дополнительных преобразований и синхронизации данных как внутри создаваемых функций, так и при их использовании:

```
[0.10, 2.1, 0.33, 1.43]:sinSwarm => [0.0998, 0.8632, 0.324, 0.9901]
```

Компилятор, анализируя тип аргумента функции `sinSwarm`, без проблем может распознать, что она принимает весь рой, а не применяется к каждому из его элементов.

2.2.1. Использование роя для упорядоченной асинхронной обработки потока

В отличие от потоков каждый даже частично сформированный рой имеет предопределенный размер. Его можно вычислить в любой момент, используя функцию `size`, сигнатура которой описывается следующим образом:

```
size << func any[*] -> int
```

Например:

```
[10,21,33,43]:size => 4
```

Нумерация элементов роя, как и вектора, начинается с единицы. Сами элементы роя формируются асинхронно. При этом поступление каждого из них сопровождается выдачей в связанный с ним оператор интерпретации сигнала, информирующего о формировании очередного значения по определенному индексу. Эти индексы можно упорядочить в порядке поступления и, следовательно, осуществить последовательную выборку отдельных элементов по ним. То есть, можно организовать итератор, делающий обход элементов роя по мере их появления. В отличие от обхода элементов потока, в которых обращение идет непосредственно за созданными элементами, в рое ключевую роль играет получение значения индекса. Для его получения предлагается использовать функцию `get`, которая имеет для роя следующую сигнатуру:

```
get << func any[*] -> int
```

То есть, возвращается индекс элемента, поступившего в рой первым.

Для перехода к следующему индексу, используется функция pop. Она возвращает ссылку на тот же рой, но уже без указания убранного индекса:

```
pop << func any[*] -> any[*]
```

Таким образом можно перебрать все элементы роя в порядке их формирования. В случае, когда через ссылку произойдет перебор всех элементов роя (в порядке их поступления), функция get возвращает нулевое значение индекса, которое, по сути, и определяет завершение обхода.

Помимо этого рой, как и поток, может использоваться для последовательной обработки асинхронно поступающих данных с сохранением порядка следования элементов на выходе. Это позволяет переписать функцию нахождения сумм квадратов синусов и косинусов роя таким образом, что она обеспечивает правильную последовательность результатов на выходе:

```
SumSin2Cos2Swarm << func X@float[*]->float[*] {
    L << X:size;
    result@float[L];
    (X, result):GetSwarmResult >> ok;
    result:ok:return
}
```

Для накопления данных функция использует дополнительное роевое хранилище result, которое заполняется с использованием принципа единственного присваивания. То есть формируется такой код, который позволяет записать данные по одному и тому же индексу не более чем один раз. При нарушении этого правила происходит прерывание выполнения программы. Через ссылку хранилище передается в функцию GetSwarmResult, обеспечивающую его заполнение, после чего полученное значение возвращается из функции SumSin2Cos2Swarm. Само вычисление осуществляется в функции GetSwarmResult:

```
GetSwarmResult << func (X@float[*],Y@float[*])->signal {
    i << X:get; // Получение индекса элемента из X
    if << (i,0):!=; // Проверка наличия элементов
    if^(
        {block {
           s << X:i:sin; Sin2 << (s,s):*; // Синус в квадрате
           c << X:i:cos; Cos2 << (c,c):*; // Косинус в квадрате
           // Вычисление текущего значения с передачей в выходной рой
           // по полученному индексу
           (Sin2,Cos2):+ -> Y[i];
           // Убирается обработанный индекс из ссылки на рой
           // и переход к обработке следующего элемента
           (X:{i:signal}:pop, Y):GetSwarmResult:break}
        },  // Занесение результата во второй рой
        // Сигнал, формируемый при завершении вычислений без заполнения,
        // если значение индекса = 0
        !
       ):return
}
```

Первоначально в данной функции вычисляется индекс первого поступившего в рой X элемента. Если значение не равно 0, то получен очередной индекс, который непосредственно используется для выборки из роя i-го элемента, после чего над ним производится вычисление суммы квадратов синуса и косинуса. Полученное значение заносится через ссылку Y на i-е место. Вычисления рекурсивно повторяются до полного заполнения хранилища result, передаваемого в данную функцию через ссылку Y.

2.2.2. Прямое обращение к элементам роя

Наряду с обработкой элементов роя в порядке их поступления возможен и непосредственный доступ по индексу. В этом случае, если элемент еще не поступил, происходит его ожидание. Во время ожидания можно инициировать выборку других элементов, используя для этого параллельно выполняемые рекурсивные вызовы. Недостатком такого подхода является возможность появления множества параллельных ветвей, ожидающих поступления данных. Однако при обработке данных, поступающих из нескольких роев данный подход облегчает синхронизацию вычислений. Сигнатура функции доступа по индексу имеет следующий вид:

```
base_function<целое> << func any[*] -> any
```

В данном случае в качестве функции используется целое число в диапазоне от 1 до размера роя. Если число не попадает в этот диапазон, то возникает прерывание в работе программы.

В качестве примера рассмотрим скалярное перемножение данных, поступающих в рои. Функция ScalMultSignal осуществляет вычисления, принимая в качестве аргументов два роя через ссылки X и Y. Помимо этого она получает ссылку R на рой, собирающий результаты, а также число, определяющее количество элементов в роях. Последнее используется в качестве индекса для обращения к элементам.

```
//  функция, непосредственно выполняющая скалярное умножение роев
ScalMultSignal << func (X@float[*], Y@float[*], R@float[*], L@int)->signal {
    if << (L,0):!=;
    if^(
        {block{
            (X:L, Y:L):* -> R[L];
            (X, Y, R, L:--):ScalMultSignal:break
        }},
        // Завершение перебора
        !
    ):return
}
```

Перемножение элементов с одинаковыми индексами осуществляется пока передаваемое значение индекса не обнулится функцией «--», формирующей значение на единицу меньше предыдущего. Рекурсивный вызов осуществляется сразу же после раскрытия задержки, охватывающей блок, независимо от того, будет или нет выполнена операция умножения.

Окончательная функция предоставляет интерфейс для взаимодействия с другими функциями:

```
// Функция, используемая для перемножения роев.
// Предполагается, что размер роев одинаков
ScalMult << func (X@float[*], Y@float[*]) -> float[*] {
    L << X:size;
    result@float[L]; // Хранилище результатов
    ok << (X, Y, R, L):ScalMultSignal;
    result:ok:return
}
```

2.2.3. Конвейеризация асинхронных потоковых вычислений

Организация вычислений функций на основе передачи между ними потоков и роев позволяет организовать взаимодействия, обеспечивающие совмещение вычислений в функциях, взаимосвязанных между собой. В качестве примера можно рассмотреть функцию векторного произведения с использованием функций скалярного произведения двух векторов `ScalMult` и нахождения суммы элементов потока `sum`:

```
VecMult << func (X@float[*], Y@float[*]) -> float {
    (X, Y):ScalMult:stream:sum:return
}
```

Данная функция принимает два роя, над которыми осуществляется выполнение скалярного произведения. По мере того, как на выходе функции `ScalMult` формируются результаты перемножения отдельных пар элементов, они поступают в поток, связанный со входом функции `sum`. Конвейеризация в данном случае формируется автоматически в зависимости от темпа поступления исходных данных и скорости выполнения операций внутри функции `VecMult`.

Заключение

Рассмотренные в работе механизмы обеспечивают поддержку нового подхода к разработке параллельных программ, позволяя описывать параллелизм с использованием асинхронных последовательно порождаемых потоков с управлением по готовности данных. При этом характеристики параллелизма зависят от темпа поступления данных. Совместное использование функций, реализованных с применением возможностей данной модели вычислений обеспечивает поддержку конвейерных вычислений. Показано, что предложенные методы описания функционально-потоковых параллельных вычислений могут быть реализованы в статически типизированном языке функционально-потокового параллельного программирования.

References

[1] A. I. Legalov, “The Usage of Asynchronous Lists within the Dataflow Model of Computations”, in *The Third Siberian School Seminar on Parallel Computations*, In Russian, 2006, pp. 113–120.

[2] C. A. R. Hoar, *Communicating Sequential Processes*, 8. Communications of the ACM, 1978, vol. 21, pp. 666–677.

[3] M. Diaz, *Petri Nets: Fundamental Models, Verification and Applications*. UK: ISTE Ltd, 2009.

[4] A. I. Legalov, “About Computation Control in Parallel System and Programming Languages”, *Nauchiy Vestnik NGTU*, vol. 18, no. 3, pp. 63–72, 2004, In Russian.

[5] A. I. Legalov, V. S. Vasiliev, and I. V. Matkovsky, “Changing Computing Management Strategies for Architecture-Independent Parallel Programming”, in *Proceedings of the XIX All-Russian Scientific Conference “Scientific Service on the Internet”*, In Russian, 2017, pp. 341–350.

[6] A. V. Redkin and A. I. Legalov, "Event Based Control of Computations for Functional Dataflow Programming", *Scientific Bulletin of Novosibirsk State Technical University*, vol. 32, no. 3, pp. 111–120, 2008, In Russian.

[7] A. I. Legalov, A. V. Redkin, and I. V. Matkovsky, "Data Driven Functional Parallel Programming with Data Coming Asynchronously", in *Parallel Computing Technologiws (PCT'2009)*, In Russian, 2009, pp. 573–578.

[8] 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.

[9] A. I. Legalov, "Functional Language for Architecture-Independent Programming", *Computation technologies*, vol. 10, no. 1, pp. 71–89, 2005, In Russian.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 27, NO. 2, 2020
journal homepage: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

Method of the Joint Clustering in Network and Correlation Spaces

A. N. Gainullina[1], A. A. Shalyto[1], A. A. Sergushichev[1]

DOI: 10.18255/1818-1015-2020-2-180-193

[1]ITMO University, 49 Kronverkskiy Prospekt, Saint Petersburg 197101, Russia.

MSC2020: 68P99, 92B99
Research article
Full text in Russian

Received June 8, 2020
After revision June 17, 2020
Accepted June 17, 2020

Network algorithms are often used to analyze and interpret the biological data. One of the widely used approaches is to solve the problem of identifying an active module, where a connected subnetwork of a biological network is selected which best reflects the difference between the two considered biological conditions. In this work this approach is extended to the case of a larger number of biological conditions and the problem of the joint clustering in network and correlation spaces is formulated.

To solve this problem, an iterative method is proposed that takes as the input graph G and matrix X, in which the rows correspond to the vertices of the graph. As the output, the algorithm produces a set of subgraphs of the graph G so that each subgraph is connected and the rows corresponding to its vertices have a high pairwise correlation. The efficiency of the method is confirmed by an experimental study on the simulated data.

Keywords: active module; clustring; gene expression; biological networks.

INFORMATION ABOUT THE AUTHORS

Anastasiia N. Gainullina	orcid.org/0000-0003-3796-2337. E-mail: anastasiia.gainullina@gmail.com PhD student.
Anatoly A. Shalyto	orcid.org/0000-0002-2723-2077. E-mail: shalyto@mail.ifmo.ru Chief researcher, professor, Doctor of Sciences.
Alexey A. Sergushichev correspondence author	orcid.org/0000-0003-1159-7220. E-mail: alserg@itmo.ru Associate professor, PhD.

Funding: Government of Russian Federation, grant 08-08.

For citation: A. N. Gainullina, A. A. Shalyto, and A. A. Sergushichev, “Method of the Joint Clustering in Network and Correlation Spaces”, *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 180-193, 2020.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 27, № 2, 2020
сайт журнала: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

Метод совместной кластеризации в графовом и корреляционном пространствах

А. Н. Гайнуллина[1], А. А. Шалыто[1], А. А. Сергушичев[1]

DOI: 10.18255/1818-1015-2020-2-180-193

[1]Университет ИТМО, Кронверкский пр. 49, г. Санкт-Петербург, 197101 Россия.

УДК 519.1
Научная статья
Полный текст на русском языке

Получена 8 июня 2020 г.
После доработки 17 июня 2020 г.
Принята к публикации 17 июня 2020 г.

Алгоритмы на графах часто используются для анализа и интерпретации биологических данных. Одним из широко используемых подходов является решение задачи поиска активного модуля, в которой в графе биологических взаимодействий выделяется связный подграф, лучше всего отражающий разницу между двумя рассматриваемыми биологическими состояниями. В настоящей работе этот подход расширяется на случай большего числа биологических состояний и формулируется задача совместной кластеризации в графовом и корреляционном пространстве.

Для решения этой задачи предлагается итеративный метод, принимающий на вход граф G и матрицу X, в которой строки соответствуют вершинам графа. На выходе алгоритм выдает набор подграфов графа G так, что каждый подграф является связным и строки, соответствующие его вершинам, обладают высокой попарной корреляцией. Эффективность метода подтверждается экспериментальным исследованием на смоделированных данных.

Ключевые слова: активный модуль; кластеризация; экспрессия генов; биологические графы.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Анастасия Наильевна Гайнуллина	orcid.org/0000-0003-3796-2337. E-mail: anastasiia.gainullina@gmail.com Аспирант.
Анатолий Абрамович Шалыто	orcid.org/0000-0002-2723-2077. E-mail: shalyto@mail.ifmo.ru Гл. науч. сотр., профессор, докт. техн. наук.
Алексей Александрович Сергушичев автор для корреспонденции	orcid.org/0000-0003-1159-7220. E-mail: alserg@itmo.ru Доцент, канд. техн. наук.

Финансирование: Правительство Российской Федерации, субсидия 08-08.

Для цитирования: A. N. Gainullina, A. A. Shalyto, and A. A. Sergushichev, “Method of the Joint Clustering in Network and Correlation Spaces”, *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 180-193, 2020.

Введение

Методы с использованием графов часто применяются в биоинформатике для интерпретации экспериментальных данных [1]. Такие методы применяются в разных контекстах: в исследованиях полногеномного поиска ассоциаций [2], метаболических процессов [3], соматических мутаций в раке [4] и других. Главная идея этих методов состоит в том, что учет известных биологических связей (например, между белками, метаболитами или другими сущностями) может повлечь более глубокое понимание данных и соответствующих биологических процессов.

Одним из часто применяемых подходов является выделение в графе биологических связей некоторого связного подграфа, лучше всего соответствующего так называемому *активному модулю*. Эта концепция впервые была представлена в работе [5], в которой авторы предложили метрику для оценки подграфов на основе данных экспрессии генов и эвристический метод *jActiveModules* для поиска наиболее оптимального подграфа. В дальнейшем эта концепция была развита в методе *BioNet* [6], в котором был предложен другой способ оценки подграфов, оптимизация которого соответствовала решению задачи поиска связного подграфа максимального веса (*Maximum Weight Connected Subgraph*, *MWCS*). Хотя эта задача является NP-полной, для нее существуют практические программы-решатели, позволяющие находить оптимальные или хорошие субоптимальные решения за небольшое время.

Однако с развитием технологий получения биологических данных эксперименты все чаще стали включать множество разнородных биологических образцов. Для таких типов экспериментов разработанные ранее методы, основанные на выделении одного активного модуля, отличающегося между двумя группами образцов, стали все менее применимы. Для них становится актуальна задача выделения нескольких активных модулей, где каждый модуль характеризуется своим профилем активности в исследуемых биологических образцах [7].

В настоящей работе рассматривается задача поиска нескольких активных модулей. Для простоты в качестве графа биологических связей используется граф белок-белковых взаимодействий, в котором вершинами являются гены, и между генами существуют ребро, если соответствующие генам белки могут взаимодействовать друг с другом в клетке. В качестве экспериментальных данных рассматриваются данные экспрессии генов, в которых каждому гену в каждом биологическом образце сопоставлено некоторое число – значение экспрессии этого гена. В таком контексте задачу поиска активных модулей можно сформулировать как задачу совместной кластеризации в графовом (на основе графа белок-белковых взаимодействий) и корреляционном (на основе таблицы экспрессии генов) пространствах.

1. Задача совместной кластеризации в графовом и корреляционном пространствах

Пусть дан граф $G = (V, E)$ порядка $n = |V|$, $V = \{1, \dots, n\}$. Пусть также дана матрица X размерности $n \times m$, где i-ая строчка матрицы соответствует i-й вершине графа. Будем называть i-ю строчку матрицы *профилем* вершины i.

Сформулируем задачу совместной кластеризации в графовом и корреляционном пространствах как поиск набора связных подграфов $S = \{S_i = (V(S_i), E(S_i))\}$ графа G таких, что:

- для каждого подграфа S_i высока попарная корреляция профилей вершин $V(S_i)_j$;
- для каждой пары подграфов S^i и S^k $(i \neq k)$ низка корреляция профилей вершин $V(S^i)_j$ и $V(S^k)_l$.

Обратим внимание, что в данном определении не требуется, чтобы подграфы S^i не пересекались. Также это определение допускает различные метрики оценки качества кластеризации, которые будут зависеть от конкретного приложения.

В настоящей работе мы будем рассматривать следующую модель. Пусть связные подграфы $A = \{A_1, \dots, A_K\}$ – истинные *активные модули*. Активность модулей в каждом из m *образцов* задается

матрицей P размерности $K \times m$. Вклад i-го гена в j-й модуль задается неотрицательной матрицей B размерности $n \times K$, при этом $B_{i,j} = 0$ как только $i \notin V(A_j)$. Тогда матрица X может быть представлена как:

$$X = B \cdot P + \varepsilon,$$

где ε – случайная матрица, соответствующая шуму.

В такой модели, задачей является по графу G и матрице X как можно более точно восстановить активные модули A. При этом K – число активных модулей – неизвестно.

2. Итеративный алгоритм совместной кластеризации

Для решения задачи совместной кластеризации в графовом и корреляционном пространствах мы предлагаем следующий алгоритм итеративной кластеризации. Алгоритм основан на идеях кластеризации с помощью алгоритма *k-means* и *EM*-алгоритма. Псевдокод алгоритма приведен на рисунке 1.

Algorithm: Network clustering

Input: Graph $G = (V, E)$ of order $n = |V|$, matrix X of size $n \times m$, initial module profiles approximation $P^{(1)}$ of size $k^{(1)} \times m$, value of *base*.

Result: Final approximation of profiles P^* of size $k^* \times m$ and a set of connected subgraphs A_i^* for $i \in 1, \ldots, k^*$ as a final approximation of active modules

for $i \in \{1, 2, \ldots\}$ **do**

- $k^{(i)} \leftarrow$ number of rows in $P^{(i)}$;
- $d_{x,y} \leftarrow 1 - \mathrm{corr}(X_x, P_y^{(i)})$ for $x \in \{1, \ldots, n\}, y \in \{1, \ldots, k^{(i)}\}$;
- $d_{x,0} \leftarrow base$ for $x \in \{1, \ldots, n\}$;
- $d'_{x,y} \leftarrow \min_{z \in \{0, \ldots, k^{(i)}\}, z \neq y} d_{x,z}$ for $x \in \{1, \ldots, n\}, y \in \{1, \ldots, k^{(i)}\}$;
- **for** $j \in \{1, \ldots, k^{(i)}\}$ **do**
  - $w_x \leftarrow -\log \frac{d_{x,y}}{d'_{x,y}}$ for $x \in \{1, \ldots, n\}$;
  - $A_j^{(i)} \leftarrow$ connected subgraph of G with maximum sum of vertex weights w;
  - $P^{(i+1)} \leftarrow$ coordinate-wise average of X_x, for $x \in V(A_j^{(i)})$ if $w_x > 0$;
- **end**
- **if** $P^{(i+1)}$ *substantially differs from* $P^{(j)}$ *for* $j \leq i$ **then**
  - **continue**
- **end**
- **if** *there are very small modules in* $A^{(i)}$ **then**
  - remove one row from $P^{(i+1)}$ that corresponds to the smallest module;
  - **continue**
- **end**
- **break**

end

Fig. 1. Proposed algorithm of clustering in network and correlation spaces

Рис. 1. Предлагаемый алгоритм совместной кластеризации в графовом и корреляционном пространствах

На i-ой итерации алгоритма выполняется два основных шага:

1. По приближению матрицы активностей модулей на предыдущей итерации $P^{(i)}$ выполняется поиск потенциальных активных модулей – подграфов $A^{(i)}$.

2. По полученным на предыдущем шаге подграфам $A^{(i)}$ выполняется коррекция матрицы активностей и получается приближение $P^{(i+1)}$ для следующего шага.

Для получения начального приближения матрицы активностей модулей $P^{(1)}$ будем использовать алгоритм кластеризации *k-medoids* для некоторого значения k, являющегося параметром. На вход этому алгоритму передается матрица d корреляционных расстояний между профилями (строчками X_i):

$$d(V_i, V_j) = 1 - \mathrm{corr}(X_i, X_j), \tag{1}$$

где corr – корреляция Пирсона. В результате работы алгоритма получается разбиение всех вершин графа на кластеры с хорошей внутренней корреляцией. Однако эти кластеры не соответствуют связным подграфам в графе G. В то же время при достаточно большом значении k, большем числа истинных активных модулей K, для всех строчек матрицы активностей модулей P в полученных кластерах будет хотя бы один медоид (вершина – центр кластера) с высокой корреляцией профиля с активностью модуля. Таким образом, в качестве начального приближения матрицы активностей модулей $P^{(1)}$ можно взять матрицу размерности $k \times m$, в которой i-ая строчка равна профилю i-го медоида.

Теперь рассмотрим шаг поиска потенциальных активных модулей по приближению матрицы активностей $P^{(i)}$ размерности $k^{(i)} \times m$.

Во-первых, определим вес, отражающий насколько хорошо вершина графа подходит к тому или иному профилю. Сначала введем корреляционное расстояние между вершиной и профилем:

$$d(V_x, P^{(i)}_y) = 1 - \mathrm{corr}(X_x, P^{(i)}_y), x \in \{1, \dots, n\}, y \in \{1, \dots, k^{(i)}\}.$$

Далее введем фиктивный нулевой профиль, расстояние до которого по определению будет всегда равно некоторой константе *base*:

$$d(V_x, P^{(i)}_0) \equiv base, x \in \{1, \dots, n\}.$$

Затем определим «референсное» расстояние до ближайшего профиля, отличного от рассматриваемого:

$$d'(V_x, P^{(i)}_y) = \min_{z \in \{0,\dots,k^{(i)}\}, z \neq y} d(V_x, P^{(i)}_z), x \in \{1, \dots, n\}, y \in \{1, \dots, k^{(i)}\}.$$

Наконец, определим искомый вес как:

$$w(V_x, P^{(i)}_y) = -\log \frac{d(V_x, P^{(i)}_y)}{d'(V_x, P^{(i)}_y)}. \tag{2}$$

Введенный вес обладает следующими свойствами:

1. Чем выше корреляция профиля вершины V_x с профилем модуля $P^{(i)}_y$, тем выше вес $w(V_x, P^{(i)}_y)$.
2. Вес $w(V_x, P^{(i)}_y)$ может быть положительным только если профиль $P^{(i)}_y$ является ближайшим к профилю вершины V_x. В частности, для заданной вершины V_x только для одного y вес $w(V_x, P^{(i)}_y)$ может быть положительным.
3. Вес $w(V_x, P^{(i)}_y)$ может быть положительным, только если корреляция между профилем вершины V_x и профилем модуля $P^{(i)}_y$ больше $1 - base$.

Таким образом, чем более положительным является вес $w(V_x, P^{(i)}_y)$, тем увереннее мы можем сказать, что вершина V_x должна принадлежать модулю для профиля $P^{(i)}_y$.

Теперь, определив вес $w(V_x, P^{(i)}_y)$ для всех вершин V_x и некоторого профиля $P^{(i)}_y$, мы можем найти такой связный подграф $A^{(i)}_y$, что суммарный вес его вершин максимален:

$$\sum_{v \in V(A^{(i)}_y)} w(v, P^{(i)}_y) \longrightarrow \max.$$

Задача поиска связного подграфа максимального веса (maximum weight connected subgraph, MWCS) является NP-полной. Однако для нее существуют несколько практических программ решателей, в том числе точных [6, 8]. Кроме того, для этой задачи есть быстрый эвристический решатель, часто находящий оптимальные или близкие к оптимальным решения [9]. В настоящей работе мы будем использовать именно этот метод, реализованный в программном пакете *mwcsr* для языка R (https://github.com/ctlab/mwcsr).

После того, как были получены подграфы $A^{(i)}$, по ним можно построить новые профили $P^{(i+1)}$. Для этого для каждого модуля усредним нормализованные (центрированные и деленные на дисперсию) значения профилей вершин модулей $A^{(i)}$ с положительным весом w. В случае, если в каком-то модуле мало положительных вершин (на практике – если одна или две), то в качестве профиля будем использовать значение с предыдущей итерации. Это позволяет не сойтись в локальный оптимум модуля.

Полученные профили $P^{(i+1)}$ затем сравниваются с профилями на предыдущих итерациях. Если профиль $P^{(i+1)}$ не совпадает ни с одним предыдущим профилем, то итерации продолжаются дальше.

В случае, если профиль $P^{(i+1)}$ совпал с каким-то из предыдущих профилей, то выполняется проверка, все ли найденные модули $A^{(i)}$ являются достаточно большими. В случае, если хотя бы один модуль имеет небольшой порядок (на практике – четыре или меньше) или небольшой диаметр (на практике – два или меньше), то выполняется процедура удаления одного модуля из рассмотрения. В качестве модуля для удаления выбирается модуль с минимальным числом вершин. Если таких модулей несколько, то выбирается один из пары наиболее скоррелированных модулей. При удалении модуля удаляется соответствующая строчка в $P^{(i+1)}$ и на единицу уменьшается число модулей $k^{(i+1)}$.

Если профиль $P^{(i+1)}$ совпал с каким-то из предыдущих профилей и все найденные модули удовлетворяют критериям по размеру, то алгоритм завершается. Последним шагом является пересчет модулей $A^{(i+1)}$ для профиля $P^{(i+1)}$. Таким образом, на выходе алгоритма получается приближение матрицы профилей активности модулей $P^* = P^{(i+1)}$ и приближение набора активных модулей $A^* = A^{(i+1)}$.

3. Экспериментальное исследование работы алгоритма

3.1. Описание симулированных данных

В настоящей работе мы будем рассматривать три типа матрицы профилей активности модулей P, соответствующие различным дизайнам биологических экспериментов. Для всех типов все биологические состояния представлены в трех повторностях – типичное число для биологических экспериментов, в которых анализируется экспрессия генов. Для упрощения сравнения между разными типами матриц во всех типах в матрицах представлены шесть биологических состояний и десять модулей.

В первой матрице P^S (рисунок 2) рассматривается простой эксперимент из шести биологических состояний. Первое состояние является контрольным – в нем активности всех модулей равны нулю. Кроме этого, есть пять других независимых состояний, и для каждого состояния есть по два независимых модуля: один активирующийся (значение активности равно единице), другой – подавляемый (значение активности равно минус одному).

В следующей матрице P^C (рисунок 3) рассматривается более сложная ситуация. В этой матрице рассматривается шесть биологических состояний и десять модулей. Каждый модуль при этом может быть активен в нескольких из биологических состояний и подавлен в остальных. Состояния, в которых активны модули, были выбраны случайным образом.

В последней матрице P^T (рисунок 4) рассматривается тип эксперимента, в котором некоторый процесс исследуется в нескольких временных точках. Как и в матрице P^S, в нем присутствует кон-

$$P^S = \begin{pmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1
\end{pmatrix}$$

Fig. 2. Matrix P for a simple experiment design with five biological conditions (excluding the control), with each condition having two corresponding modules: one that gets activated and one that gets inhibited

Рис. 2. Матрица P для простого типа эксперимента с пятью биологическими состояниями, не считая контрольного, где каждому состоянию соответствуют два модуля: активирующийся и подавляющийся

$$P^C = \begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0
\end{pmatrix}$$

Fig. 3. Matrix P for a complex experiment design with six biological conditions, with each module being active in a certain subset of the conditions

Рис. 3. Матрица P для сложного типа эксперимента с шестью биологическими состояниями, где каждый модуль активен в некотором подмножестве состояний

трольное состояние. Еще пять состояний соответствуют последовательным временным точкам. Для каждой временной точки присутствует модуль, который в ней начинает активироваться, и модуль, который начинает подавляться. Для такого профиля активности может усложниться разделение модулей, так как модули соответствующие близким временным точкам имеют высокую корреляцию профилей.

В качестве графа G во всех экспериментах рассматривался граф белок-белковых взаимодействий, используемый в пакете *BioNet*. Граф состоит из 2034 вершин и 7756 ребер.

Активные модули A_i генерировались следующим образом. Во всех случаях число вершин в модуле выбиралось случайно перестановкой множества $\{20, 40, \dots, 200\}$. После того, как порядок модуля был выбран, модуль выбирался равномерным случайным образом из всех связных подграфов такого порядка. Для генерации случайных подграфов использовался пакет *mcmcRanking*.

После того, как выбраны модули A_i, задавалась матрица B вклада модуля в профили вершин. Для всех ненулевых элементов $B_{i,j}$ (соответствующих вершинам i, входящим в модуль A_j) значение случайно выбиралось из экспоненциального распределения $\mathrm{Exp}(\lambda)$ с параметром $\lambda = 1$.

$$P^T = \begin{pmatrix}
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1
\end{pmatrix}$$

Fig. 4. Matrix P for a time course experiment design where each module either gets activated or inhibited at certain time point and keeps its state till the end of the experiment

Рис. 4. Матрица P для эксперимента с несколькими временными точками, в котором каждый модуль либо активируется, либо подавляется в некоторый момент времени и сохраняет свое состояние до конца эксперимента

Наконец, шум ε генерировался из нормального распределения $\mathcal{N}(0, \sigma^2)$. Значение среднеквадратичного отклонения σ являлось варьируемым параметром – большие значения среднеквадратичного отклонения усложняют задачу поиска модулей.

3.2. Базовые методы для сравнения

Для сравнения использовалось три базовых метода.

Первый метод выполняет кластеризацию *k-medoids* по матрице корреляционных расстояний (1). Значение k может варьироваться. Результатом является набор из k непересекающихся множеств – найденных кластеров.

Второй метод также выполняет кластеризацию, но методом *WGCNA* (weighted gene correlation network analysis) [10]. Аналогично предыдущему, для кластеризации этот метод использует корреляционное расстояние. Одной из особенностей метода является то, что в этом методе заранее не задается число кластеров, а его работа может регулироваться другими параметрами. Другой особенностью является то, что он способен выделить «мусорный» кластер, состоящий из вершин, которые не относятся ни к какому из «настоящих» кластеров с хорошей внутренней корреляцией. В качестве результата для сравнения используется набор из выделенных методом *WGCNA* непересекающихся кластеров, за исключением «мусорного».

Третьему методу (будем называть его *nearest*) передается истинная матрица профилей P и параметр *base*. Метод вычисляет вес $w(V_x, P_y^{(i)})$ по формуле (2). Метод возвращает K кластеров, где в i-й кластер входят все вершины V_x, для которых $w(V_x, P_y^{(i)}) > 0$.

3.3. Сравнение методов получения стартовых приближений

В первом эксперименте было проведено сравнение способов получения стартовых приближений. В сравнении участвовало четыре метода:

1. Метод *k-means* со значениями k из множества $\{16, 24, 32, 64, 96\}$. В качестве расстояния использовалось евклидово расстояние. Дополнительная модификация значений экспрессии не проводилась.
2. Метод *k-medoids* со значениями k из множества $\{16, 24, 32, 64, 96\}$. В качестве метрики использовалось корреляционное расстояние.
3. Метод *WGCNA*. В качестве финального профиля использовалось покоординатное усреднение профилей, входящих в соответствующий кластер.

4. Метод *all* – метод для сравнения, который возвращал все строки исходной матрицы X.

Все методы были запущены на всех трех типах исходных матриц P. Среднеквадратичное отклонение для шума σ выбиралось из множества $\{0.25, 0.5, 1\}$. Каждый эксперимент проводился пять раз для разных исходных значений состояния генератора случайных чисел.

Для оценки качества получаемых стартовых приближений использовалась следующая процедура. Вычислялись попарные корреляции между строчками истиной матрицы P и стартовыми приближениями S, полученными некоторым методом. Затем для каждой строки матрицы P выбиралось максимальное значение корреляции, таким образом, получалось значение того, как хорошо соответствующая строка P представлена в матрице S. Наконец, вычислялись две суммирующие метрики: *corAvg* – усредненное значение корреляции для всех строк P, и *corMin* – минимальное значение корреляции.

На рисунке 5 приведены результаты сравнения. Для упрощения визуализации данные по разным типам матриц P объединены, так как показывают похожее поведение. В соответствии с ожиданиями, с увеличением среднеквадратичного шума все методы хуже справляются с восстановлением исходных профилей. Методы *k-means* и *k-medoids* показывают похожие значения для одинаковых значений k, и эти результаты улучшаются с увеличением k, однако на значении $k = 64$ уже наступает насыщение. Метод $WGCNA$ почти всегда проигрывает методам *k-medoids* и *k-means* с $k \geq 32$, кроме случая большого шума ($\sigma = 1$), где результаты похожи.

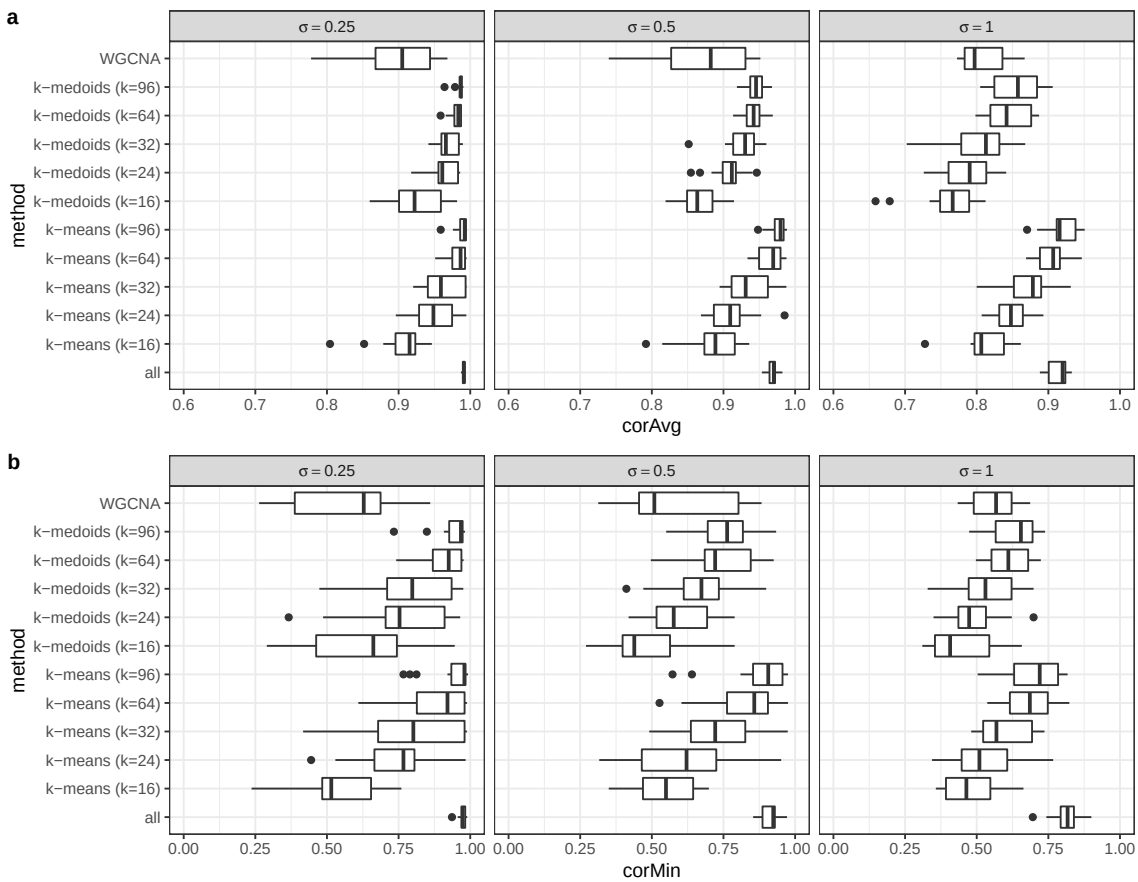


Fig. 5. Comparison of different start profiles generation methods by correlation with true values of P

Рис. 5. Сравнение различных способов получения стартовых профилей по корреляции с истинными профилями P

Также отметим, что для методов *k-means* и *k-medoids* для достижения хороших показателей требуются значения k в несколько раз больше истинного числа модулей (десяти). Таким образом, актуальной является задача разработки метода для восстановления матрицы P даже без ограничений на связность модулей в графе G.

3.4. Старт с истинных значений

В следующем эксперименте было проведено исследование качества работы предлагаемого метода кластеризации при старте с истинных значений P. Как и в предыдущем эксперименте, запуск проводился на всех трех матрицах P^S, P^C и P^T. Среднеквадратичное отклонение для шума σ выбиралось из множества $\{0.25, 0.5, 1\}$. Каждый эксперимент проводился пять раз для разных исходных значений состояния генератора случайных чисел.

Для каждого сгенерированного набора предлагаемый метод (*net-clust*) запускался со значениями *base* из множества $\{0.2, 0.3, 0.4, 0.5, 0.6\}$. Текущая матрица P передавалась в качестве стартового значения $P^{(1)}$. Для сравнения использовался метод *nearest*, описанный выше, которому также передавалась матрица P и значение *base*.

Оценка качества результатов выполнялась следующим образом. Во-первых, задача рассматривалась как задача классификации вершин на те, которые принадлежат хотя бы одному модулю, и те, которые не принадлежат. В этом случае можно вычислить метрики точности (*precision*) и полноты (*recall*). Однако эти метрики не отражают, насколько хорошо вершины разделяются на отдельные модули. Чтобы это учесть, для каждого найденного модуля вычислялась максимальная доля его вершин, полностью совпадающих с одним из истинных модулей A. Усредненное значение этих долей обозначалось как метрика *average module consistency* – чем ближе ее значение к единице, тем лучше.

Результаты анализа метрик точности и полноты представлены на рисунке 6 (панели *a* и *b*, соответственно). Как и в предыдущем эксперименте, поведение метрик не сильно зависело от типа матрицы P, поэтому данные приведены в агрегированном по всем типам виде. Результаты показывают, что добавление ограничения на связность влияет на результаты, но не очень сильно. Для больших значений *base* метод *net-clust* дает большую точность, но меньшую полноту.

Также на рисунке 6*c* приведен анализ метрики *average module consistency*. В целом, поведение этой метрики достаточно хорошо повторяет поведение метрики точности, однако на ней наблюдается выход предлагаемого метода *net-clust* на плато при значениях $base \leq 0.4$ и «провале» относительно базового метода *nearest*.

3.5. Исследование работы алгоритма с разных способов получения начальных приближений

Затем было исследовано, насколько влияет способ получения начальных приближений матрицы P на конечный результат работы алгоритма. Для получения начальных приближений использовались методы *k-means* и *k-medoids* со значениями $k = 32$ и $k = 64$. Исследование проводилось только для $P = P^S$. Среднеквадратичное отклонение для шума σ выбиралось из множества $\{0.25, 0.5, 1\}$. Для каждой пары матрицы P и выбранного уровня шума σ генерировалось по три набора данных для разных исходных значений состояния генератора случайных чисел.

Результаты исследования показали, что результат работы алгоритма не сильно зависит от того, как выбирался начальный набор. Так как время работы алгоритма было значительно больше для $k = 64$ по сравнению с $k = 32$ и метод *k-medoids* при $k = 32$ показал чуть лучшие результаты по сравнению с методом *k-means* для того же k, в дальнейшем сравнении для начального приближения использовался только метод *k-medoids* с $k = 32$.

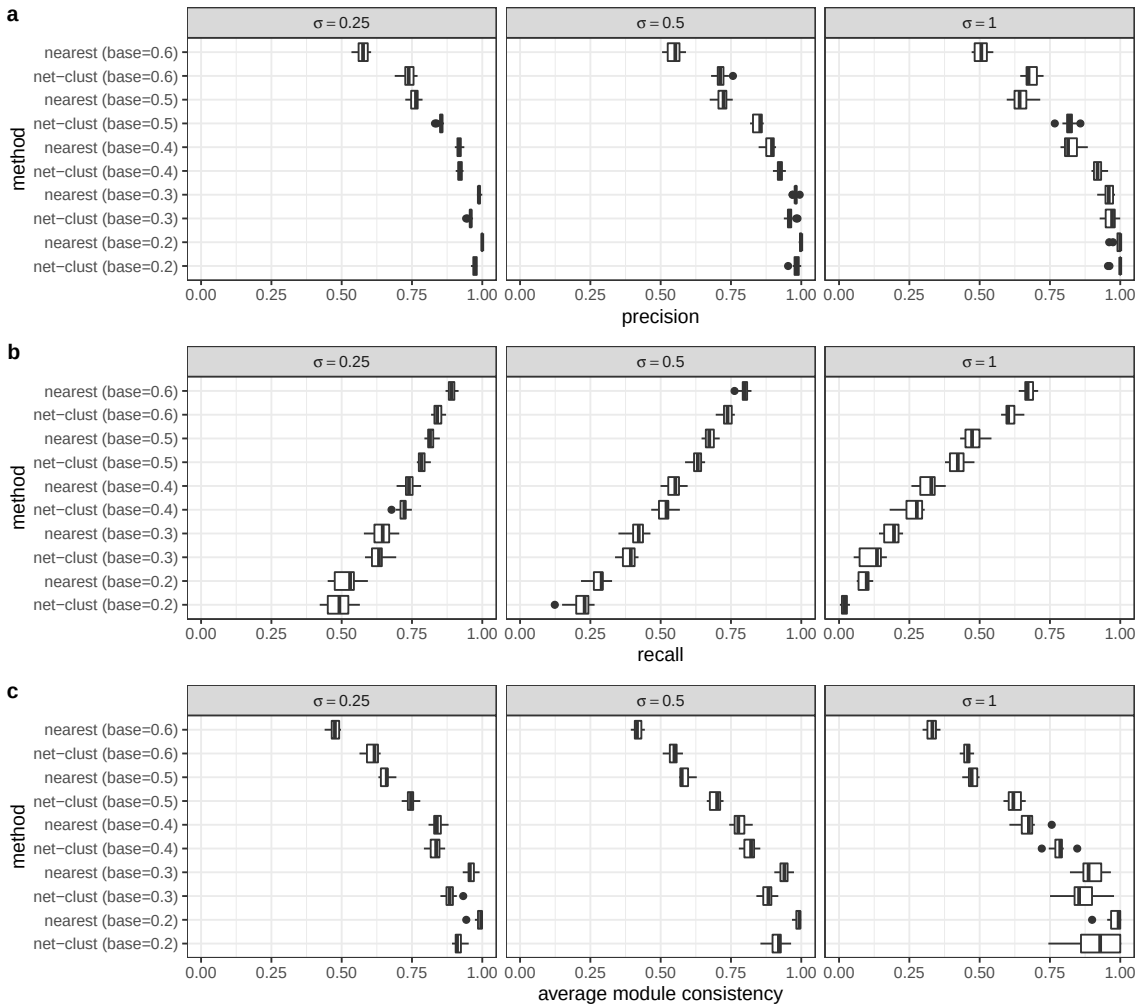


Fig. 6. Analysis of results of method *net-clust* as run with true values of P as a start approximation and compared with *nearest* method as a baseline

Рис. 6. Анализ результатов предлагаемого метода *net-clust* по сравнению с базовым методом *nearest* при старте с истинного значения матрицы P

3.6. Итоговое сравнение алгоритма с базовыми методами

Последний эксперимент заключался в запуске предлагаемого алгоритма от начала до конца и сравнении его с базовыми методами. Исходные данные для эксперимента генерировались для всех трех матриц P^S, P^C и P^T. Как и в предыдущих экспериментах, среднеквадратичное отклонение для шума σ выбиралось из множества $\{0.25, 0.5, 1\}$. Для каждой пары матрицы P и выбранного уровня шума σ генерировалось по три набора данных для разных исходных значений состояния генератора случайных чисел.

Предлагаемый метод запускался со стартовых значений, полученных методом *k-medoids* с параметром $k = 32$. Параметр *base* выбирался из множества $\{0.3, 0.4, 0.5\}$. Для сравнения использовался метод *k-medoids*, также с параметром $k = 32$, метод *WGCNA* и метод *nearest* с таким же выбором параметра *base*.

Результаты оценки качества алгоритмов представлены на рисунке 7. Как и ранее, результаты для разных типов матрицы P отличаются незначительно, поэтому приведены в агрегированном виде.

Из этого эксперимента видно, что предлагаемый метод позволяет достигать высокой точности в определении модулей при достаточно неплохой полноте.

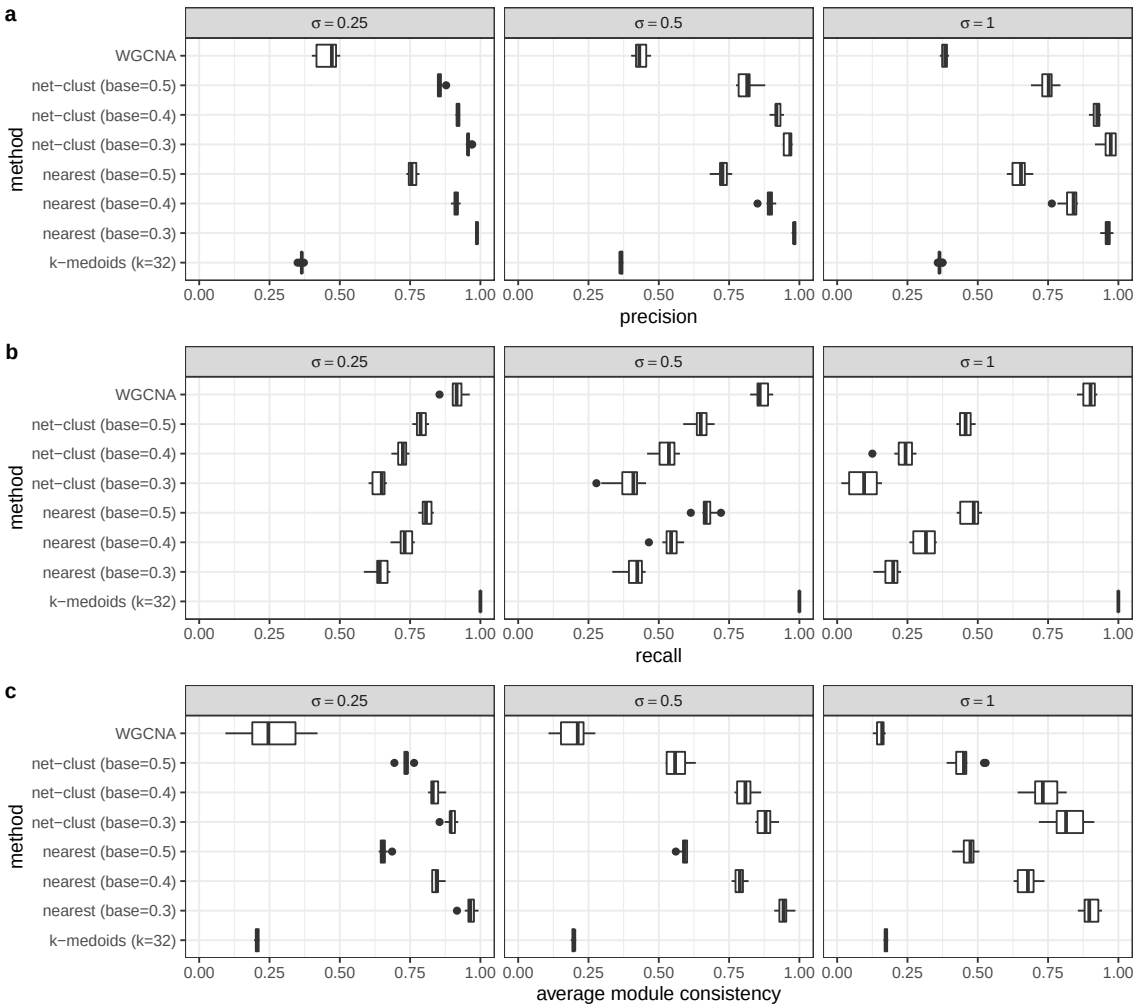


Fig. 7. Analysis of results of method *net-clust* compared with baseline methods

Рис. 7. Анализ результатов предлагаемого метода *net-clust* по сравнению с базовыми методами

Отдельно важно отметить, что качество работы предлагаемого алгоритма сравнимо с качеством базового метода *nearest*, запускаемого с теми же значениями *base*. При этом методу *nearest* на вход передается истинная матрица P, а предлагаемый метод *net-clust* выводит ее автоматически.

Заключение

В работе рассматривается задача поиска активных модулей в графах белок-белковых взаимодействий по данным экспрессии генов. Математически, эта задача была сформулирована как задача совместной кластеризации в графовом и корреляционном пространствах.

В настоящей работе впервые сформулирована математическая модель этой задачи и предложены метрики оценки качества решений. В рамках этой формулировки для решения задачи был предложен метод *net-clust* основанный, с одной стороны, на идеях решения задачи поиска одного активного модуля, с другой – на идеях алгоритма *k-means*. В методе последовательно приближается

набор профилей искомых кластеров (центров) и находятся связные подграфы, профили вершин которых хорошо коррелируют с центрами.

Так как для решения этой задачи отсутствуют точные аналоги, для оценки качества метода были сформулированы несколько базовых методов. В качестве базовых, были рассмотрены как методы кластеризации, применяющиеся для анализа экспрессии генов: *k-medoids* и *WGCNA*, так и метод, использующий информацию об истинных значения центров кластеров – не применимый на практике, но позволяющий оценить качество оптимального решения. Все базовые методы не требуют связности кластеров в заданном графе биологических связей, в отличие от предлагаемого метода *net-clust*.

Экспериментальное исследование было проведено на симулированных данных, для которых известен правильный ответ и, таким образом, на которых можно сравнить качество предлагаемого метода с базовыми методами. Исследование показало, что для получения начальных приближений профилей кластеров подходят методы *k-medoids* и *k-means* при значениях k в несколько раз превышающих истинное число модулей. При этом дальнейшее увеличение k не приводит к улучшению результатов, но увеличивает время работы метода.

Сравнение с практическими методами *k-means* и *WGCNA* показало, что метод *net-clust* имеет значительно более высокую точность. С другой стороны, предлагаемый метод *net-clust* достигает параметров качества сравнимых с методом, принимающим на вход истинные значения центров кластеров, что означает, что качество метода *net-clust* близко к оптимально возможному для этой задачи.

Таким образом, было показано, что задача поиска активных модулей может быть сформулирована как задача совместной кластеризации в графовом и корреляционном пространствах, которая, в свою очередь, может быть решена близко к оптимальности с помощью предлагаемого метода *net-clust*.

References

[1] K. Mitra, A. R. Carvunis, S. K. Ramesh, and T. Ideker, “Integrative approaches for finding modular structure in biological networks”, *Nat. Rev. Genet.*, vol. 14, no. 10, pp. 719–732, 2013.

[2] E. J. Rossin *et al.*, “Proteins encoded in genomic regions associated with immune-mediated disease physically interact and suggest underlying biology”, *PLoS Genet.*, vol. 7, no. 1, e1001273, 2011.

[3] A. K. Jha, S. C. Huang, A. Sergushichev, V. Lampropoulou, Y. Ivanova, E. Loginicheva, K. Chmielewski, K. M. Stewart, J. Ashall, B. Everts, E. J. Pearce, E. M. Driggers, and M. N. Artyomov, “Network integration of parallel metabolic and transcriptional data reveals metabolic modules that regulate macrophage polarization”, *Immunity*, vol. 42, no. 3, pp. 419–430, 2015.

[4] M. D. Leiserson *et al.*, “Pan-cancer network analysis identifies combinations of rare somatic mutations across pathways and protein complexes”, *Nat. Genet.*, vol. 47, no. 2, pp. 106–114, 2015.

[5] T. Ideker, O. Ozier, B. Schwikowski, and A. F. Siegel, “Discovering regulatory and signalling circuits in molecular interaction networks”, *Bioinformatics (Oxford, England)*, vol. 18 Suppl 1, S233–S240, 2002.

[6] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Müller, “Identifying functional modules in protein-protein interaction networks: an integrated exact approach.”, *Bioinformatics (Oxford, England)*, vol. 24, no. 13, pp. i223–31, 2008, ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btn161.

[7] M. N. Artyomov, A. Sergushichev, and J. D. Schilling, “Integrating immunometabolism and macrophage diversity”, *Semin. Immunol.*, vol. 28, no. 5, pp. 417–424, Oct. 2016.

[8] A. A. Loboda, M. N. Artyomov, and A. A. Sergushichev, "Solving Generalized Maximum-Weight Connected Subgraph Problem for Network Enrichment Analysis", in *Algorithms in Bioinformatics: 16th International Workshop, WABI 2016, Aarhus, Denmark, August 22-24, 2016. Proceedings*. Cham: Springer International Publishing, 2016, pp. 210–221, ISBN: 978-3-319-43681-4.

[9] E. Álvarez-Miranda and M. Sinnl, "A Relax-and-Cut framework for large-scale maximum weight connected subgraph problems", *Computers & Operations Research*, vol. 87, pp. 63–82, 2017.

[10] P. Langfelder and S. Horvath, "WGCNA: an R package for weighted correlation network analysis", *BMC Bioinformatics*, vol. 9, p. 559, 2008.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 27, NO. 2, 2020
journal homepage: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

"VTMine for Visio": Graphical Tool for Modeling in Process Mining

S. A. Shershakov[1]

DOI: 10.18255/1818-1015-2020-2-194-217

[1]National Research University Higher School of Economics, 20 Myasnitskaya st., Moscow 101000, Russia.

MSC2020: 68U35, 93A30
Research article
Full text in Russian

Received May 25, 2020
After revision June 5, 2020
Accepted June 10, 2020

Process-Aware Information Systems (PAIS) is a special class of the IS intended for the support the tasks of initialization, end-to-end management and completion of business processes. During the operation such systems accumulate a large number of data that are recorded in the form of the event logs. Event logs are a valuable source of knowledge about the actual behavior of a system. For example, there can be found information about the discrepancy between the real and the prescribed behavior of the system; to identify bottlenecks and performance issues; to detect anti-patterns of building a business system. These problems are studied by the discipline called "Process Mining".

The practical application of the process mining methods and practices is carried out using the specialized software for data analysts. The subject area of the process analysis involves the work of an analyst with a large number of graphical models. Such work will be more efficient with a convenient graphical modeling tool. The paper discusses the principles of building a graphical tool "VTMine for Visio" for the process modeling, based on the widespread application for business intelligence Microsoft Visio. There are presented features of the architecture design of the software extension for application in the process mining domain and integration with the existing libraries and tools for working with data. The application of the developed tool for solving various types of tasks for modeling and analysis of processes is demonstrated on a set of experimental schemes.

Keywords: process modeling; process mining; experiment models; graphical tool; experiments automation.

INFORMATION ABOUT THE AUTHORS

Sergey A. Shershakov | orcid.org/0000-0001-8173-5970. E-mail: sshershakov@hse.ru
correspondence author | researcher.

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

For citation: S. A. Shershakov, ""VTMine for Visio": Graphical Tool for Modeling in Process Mining", *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 194-217, 2020.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 27, № 2, 2020
сайт журнала: www.mais-journal.ru

COMPUTING METHODOLOGIES AND APPLICATIONS

“VTMine for Visio”: инструмент графического моделирования в области Process Mining

С. А. Шершаков[1]

DOI: 10.18255/1818-1015-2020-2-194-217

[1]Национальный исследовательский университет «Высшая школа экономики», Мясницкая ул., д. 20, г. Москва, 101000 Россия .

УДК 519.682.6+004.94
Научная статья
Полный текст на русском языке

Получена 25 мая 2020 г.
После доработки 5 июня 2020 г.
Принята к публикации 10 июня 2020 г.

Процессно-ориентированные информационные системы (ПОИС) – специальный класс ИС для поддержки задач по инициализации, сквозному управлению и завершению бизнес-процессов. В процессе функционирования такие системы накапливают большое число данных, которые записываются в виде журналов событий. Журналы событий являются ценным источником знаний о реальном поведении системы. Например, в них можно обнаружить информацию о несоответствии реального и желаемого поведения системы; определить узкие места и проблемы с производительностью; детектировать анти-паттерны построения бизнес-системы. Изучением этих задач занимается дисциплина «Извлечение и анализ моделей процессов» (Process Mining).
Практическое применение методов и практик Process Mining осуществляется с помощью специализированного программного обеспечения (ПО) для аналитиков данных. Предметная область анализа процессов подразумевает работу аналитика с большим числом графических моделей. Такая работа будет более эффективной при наличии удобного инструмента графического моделирования. В настоящей работе рассматриваются принципы построения графического инструмента «VTMine for Visio» моделирования процессов на базе распространенного приложения для бизнес-аналитики Microsoft Visio. Приводятся особенности проектирования архитектуры программного расширения для применения в области Process Mining и интеграции с существующими библиотеками и инструментами для работы с данными. Применение разработанного приложения для решения различного вида задач по моделированию и анализу процессов демонстрируется на наборе схем экспериментов.

Ключевые слова: моделирование процессов; извлечение и анализ моделей процессов; модели экспериментов; графический инструмент; автоматизация экспериментов.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Сергей Андреевич Шершаков
автор для корреспонденции

orcid.org/0000-0001-8173-5970. E-mail: sshershakov@hse.ru
научный сотрудник.

Финансирование: Работа выполнена в рамках Программы фундаментальных исследований НИУ ВШЭ.

Для цитирования: S. A. Shershakov, ““VTMine for Visio”: Graphical Tool for Modeling in Process Mining”, *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 194-217, 2020.

Введение

Последние годы обозначились активным развитием информационных систем (ИС), которые проникли во все сферы человеческой деятельности. Это связано с повышением производительности оборудования, снижением его стоимости, появлением новых инструментов и подходов к разработке ИС и развитием кадрового рынка специалистов в информационных технологиях (ИТ).

Все это повлияло на ускорение окружающих нас бизнес-процессов и увеличение объемов связанных с ними данных. Для поддержки таких процессов было разработано большое количестве ИС, которые сформировали отдельный класс — *процессно-ориентированных информационных систем* (ПОИС). Они представляют собой сложные распределенные программные комплексы, принимающие на себя задачи по инициализации, сквозной поддержке и завершению бизнес-процессов. В процессе функционирования такие системы накапливают большое число побочных данных, которые записываются в *логах*. Логи могут рассматриваться с технической точки зрения (*системные логи*) или с точки зрения *следов поведения системы* относительно событий бизнес-процесса (*логи событий*). Последние являются ценным источником знаний о реальном поведении системы. Сравнивая реальное поведение с ожидаемым, можно обнаружить массу полезной информации: несоответствие реального и желаемого поведения системы; узкие места и проблемы с производительностью; анти-паттерны построения бизнес-системы и т. д.

Изучением этих задач занимается дисциплина «*извлечение и анализ моделей процессов*» (англ. *Process Mining*, *PM*) [1]. Субъектами этой дисциплины являются бизнес-аналитики, владельцы процессов, исследователи данных. Ее основными объектами являются бизнес-процессы. Процессы представляются различными абстракциями в виде статистических, графических и других видов моделей, а также конкретными выполненными экземплярами процессов, записанными в логах событий.

Несмотря на то, что извлечение процессов является относительно новым направлением, оно уже активно применяется при моделировании и анализе бизнес-процессов [2] в менеджменте, при разработке программного обеспечения [3, 4], в управлении технологическими процессами, в медицине [5, 6].

Практическое применение методов и практик Process Mining осуществляется с помощью специализированного программного обеспечения (ПО) для аналитиков данных. ПО может носить исследовательский (академический) характер, быть ориентировано на индустриальное применение, либо совмещать в себе две эти функции. В зависимости от назначения, к ПО могут предъявляться различные требования: по фунциональному наполнению, производительности, модульности, наличии интеграции с существующими инструментами бизнес-аналитики и т.д.

Предметная область анализа процессов подразумевает работу аналитика с большим числом графических моделей. Такая работа будет более эффективной при наличии удобного инструмента графического моделирования. В настоящей работе рассматриваются принципы построения графического инструмента *VTMine for Visio* моделирования процессов на базе распространенного приложения для бизнес-аналитики Microsoft Visio [7]. Показано, как можно расширить функционал существующего инструмента для применения в области Process Mining. Демонстрируется применение разработанного приложения для решения различного вида задач по моделированию и анализу процессов.

Существующие решения

Программные инструменты, связанные с областью Process Mining, которые наиболее часто упоминаются в литературе [8, 9], а также инструменты, получившие развитие в последнее время, можно разбить на условные группы по следующему принципу.

К первой группе относятся инструменты, представляющие графический пользовательский интерфейс (GUI) для моделирования и анализа. К этой группе относятся: *ProM* [10], *RapidProM* [11], *DPMine/P* [12], *Disco* [13], *Celonis* [14], *Minit* [15]. Ко второй группе относятся инструменты, реализованные в виде утилит командной строки: *Petrify* [16, 17], *Rbminer* [18], *genet* [19]. К третьей группе относятся инструменты, реализованные в виде пакетов-расширений к скриптовым языкам Python — *PMLab* [20], *PM4Py* [21] и R — *bupaR* [22].

Из представленных инструментов поддержку графического управления потоками задач (workflow) имеют только два инструмента — *RapidProM* и *DPMine/P*, оба являются надстройками над инструментом *ProM*.

Остальная часть статьи организована следующим образом. В разделе 1 рассматривается объектная модель инструмента MS Visio и ее особенности, позволяющие выполнять расширение этого инструмента для задач Process Mining. В разделе 2 обсуждаются особенности проектирования архитектуры приложения для графического моделирования с учетом специфики моделирования процессов. В частности, в разделе 2.2 рассматривается подсистема моделирования экспериментов *DPMine*, основанная на одноименном графическом языке. В разделе 3 представлены примеры моделей экспериментов, реализованных с помощью разработанного инструмента. Наконец, в заключении подводятся итоги проделанной работы и рассматриваются направления для будущей работы.

1. Объектная модель инструмента MS Visio

Все компоненты приложения MS Visio, включая элементы пользовательского интерфейса, представляются в виде COM-классов с экспортированными интерфейсами и доступны для взаимодействия с программными расширениями, разработанными в виде макросов с помощью интерпретируемого языка сценариев Visual Basic for Application (VBA), либо в виде расширений Visio Add-In на языках программирования платформы .NET (C#, C++/CLI и др.).

Основными компонентами, используемыми для представления документа Visio, являются следующие классы: `Document`, `Page` и `Shape`[1]. Объект класса `Document` представляет один загруженный в память Visio-документ, который может быть файлом с *графической моделью* или рисунком (расширения .vsd, .vdx), файлом с *коллекцией элементов-заготовок* (stencils, расширения .vst и .vtx) либо *файлом-шаблоном* (расширение .vst или .vtx). Документы состоят из «плоского» списка страниц, каждая из которых представляется объектом класса `Page`. Каждая страница содержит коллекцию фигур, имеющих графическое представление. В объектной модели каждой фигуре соответствует объект класса `Shape`. Фигуры на странице упорядочены по z-оси, что определяет их перекрытие друг другом. Фигура может быть *группировочной* (группой фигур), что позволяет задавать иерархию фигур. Каждая группа рассматривается в качестве одной фигуры — на странице, где она лежит, либо внутри другой группы.

1.1. Семантические атрибуты компонентов Visio

Все три ключевых компонента Visio-документа являются *атрибутированными*, то есть их *внешний вид*, *поведение* и взаимодействие с другими компонентами определяется набором атрибутов, приписанных конкретному объекту. Атрибуты в Visio представляются в виде специальной *таблицы свойств* (property sheet). На уровне объектной модели работа с атрибутами осуществляется путем обращения к соответствующим свойствам, полям и методам — в зависимости от вида Visio-расширения и языка программирования. Редактирование некоторых атрибутов доступно с помощью специального визуального редактора (property sheet editor).

[1] Все эти типы определены в пространстве имен `Microsoft.Office.Interop.Visio`

Атрибуты фигур, страниц и документов сгруппированы по функциональному назначению. Например, секция *Shape Transform* содержит атрибуты некоторой фигуры, определяющие ее геометрические размеры и позицию на родительском компоненте (странице или группе фигур); одна или несколько секций *Geometry X* определяют набор графических примитивов, используемых для отрисовки фигур; секции *Line Format*, *Fill Format* и *Text Block Format* определяют такие характеристики, как цвет линии, заливки, тип штриха, параметры шрифта и др.

Значение атрибутов объекта может задаваться как с помощью констант — числовых или символьных литералов, так и с помощью формул-выражений, включающих ссылки на другие атрибуты этого или другого объекта. Это позволяет взаимосвязывать значения атрибутов без необходимости осуществления программного контроля. В общем случае в качестве значения атрибута может выступать строка достаточно большого размера. Использование длинных строк позволяет сериализовать объект произвольной структуры, тип данных которого не является одним из базовых (целое или с плавающей точкой число, список-перечисление, булевый тип, строка и некоторые др.). Значения атрибутов, приписанных любому объекту, автоматически сериализуются и записываются как составная часть документа при сохранении его в долговременной памяти.

В рамках данной работы наибольший интерес представляют следующие важнейшие секции: *Shape Data* и *User-defined Cells*. Эти секции позволяют приписать соответствующему объекту дополнительные атрибуты заданных типов данных, которые будут отождествлены с объектом и также автоматически будут сохраняться как часть документа и восстанавливаться при его загрузке. Однако в отличие от стандартных атрибутов использование данных атрибутов является *опциональным* и определяется соответствующим программным расширением.

Различие между секциями *Shape Data* и *User-defined Cells* заключается в их назначении. Атрибуты секции *Shape Data* доступны для обращения через управляющий элемент пользовательского интерфейса *Shape Data Pane*. С помощью этой панели пользователь задает значения параметров, ассоциированных с конкретной фигурой. С этой целью каждый атрибут, описанный в данной секции, содержит помимо уникального имени ряд параметров, определяющих его визуальное представление: тип данных, способ форматирования, отображения и др. Атрибуты такого типа совмещают в себе одновременно элементы «модели, представления и контроллера» при рассмотрении их с точки зрения паттерна проектирования «модель-вид-представление» (MVC). Примерами таких атрибутов являются атрибуты «стоимость», «владелец», «дата», приписываемые фигурами модели блок-схемы; атрибуты «категории», «исполнители» блока «задача» модели BPMN и др.

В отличии от секции *Shape Data* секция *User-defined Cells* содержит набор атрибутов, которые не имеют визуального представления, и соответствуют только элементу «модель» паттерна MVC. Такие атрибуты определяются только своим именем и значением, тип которого является универсальным и требует соответствующей интерпретации при обращении к нему программным образом. Атрибуты секции *User-defined Cells* не отображаются в стандартной панели *Shape Data Pane*, однако доступны программно, а также посредством непосредственного обращения к таблице свойств соответствующего объекта. Назначением атрибутов такого типа является т. н. «семантическое атрибутирование», то есть добавление к некоторому объекту (фигуре или странице) произвольных дополнительных атрибутов, не определенных типами данных этих объектов (`Shape` и `Page` соответственно). Данные атрибуты используются для задания таких свойств объектов, непосредственный доступ к которым пользователя приложения не требуется либо нежелателен. Примерами моделей, использующих в основном атрибуты секции *User-defined Cells*, являются модели семейства UML, доступные в стандартной поставке Visio. С помощью таких атрибутов осуществляет управление поведением таких элементов UML-моделей, как *Class*, *Relation* и др. Задание свойств этих объектов осуществляется с помощью специальных визуальных управляющих окон, содержащих сложные специализированные элементы управления, комбинация значений которых записыва-

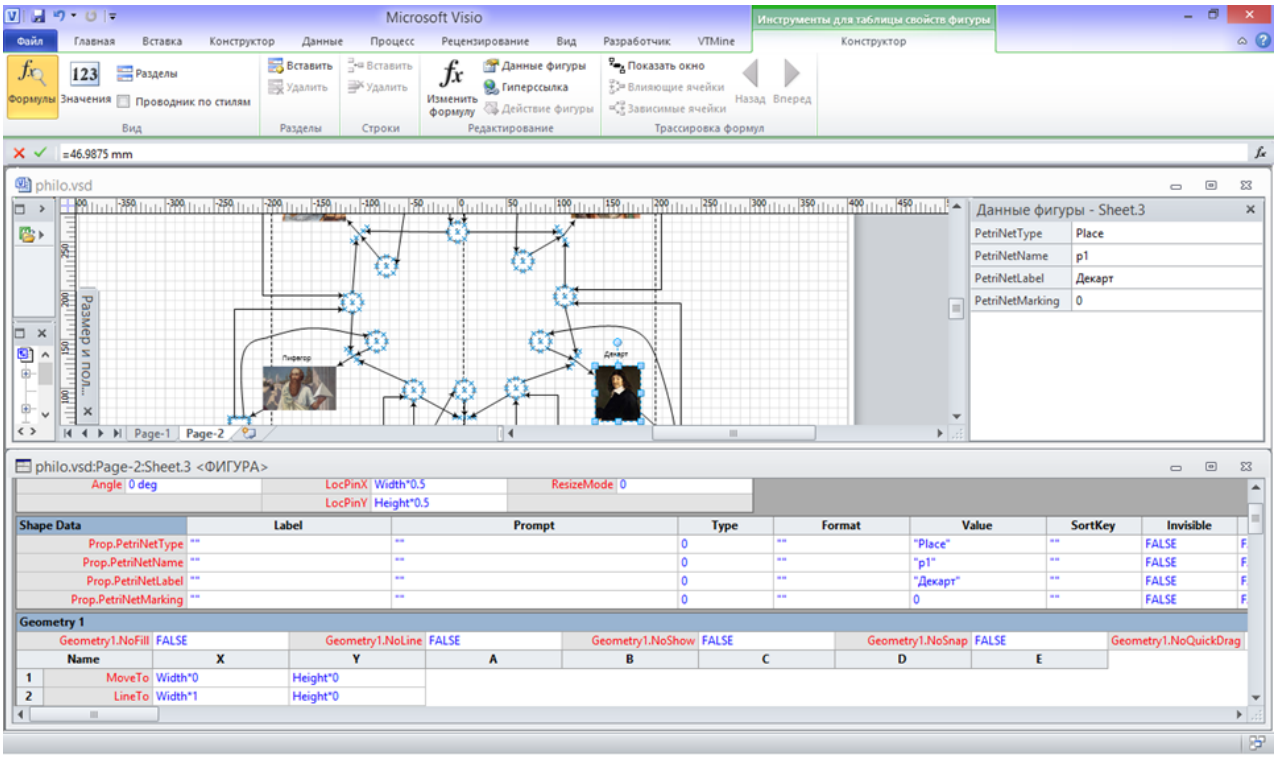


Fig. 1. Graphical representation of a Petri net modeling *the Dining Philosopher Problem*

Рис. 1. Графическое представление сети Петри, моделирующей *задачу об обедающих философах*

ется в соответствующий атрибут или набор атрибутов программно с помощью соответствующего программного расширения (add-in).

Программное обращение к обоим типам «семантических» атрибутов осуществляется практически идентично с помощью уникального имени атрибута и префикса, определяющего секцию: `User` для атрибутов секции *User-defined Cells* и `Prop` для атрибутов секции *Shape Data*. Далее мы будем ссылаться на них как на *пользовательские атрибуты закрытого* и *открытого* типов соответственно.

Использование *пользовательских атрибутов* позволяет рассматривать документ Visio не только в качестве векторного рисунка, но и наделяет его семантическими свойствами. На рисунке 1 изображен пример известной задачи «обедающих философов» [23, 24], которая моделируется с помощью *сети Петри*. Из иллюстрации видно, что рисунок модели содержит несколько изображений известных ученых и философов, каждое из которых является графическим примитивом Visio «растровый рисунок». Эти графические примитивы связаны с другими графическими примитивами (черными вытянутыми прямоугольниками, кругами, квадратами) с помощью направленных коннекторов. Вместе они образуют графическое представление сети Петри, в которой круги традиционно обозначают позиции, квадраты и прямоугольники — переходы, а направленные коннекторы — дуги.

Элементы пользовательских свойств этих фигур содержат именованные атрибуты, определяющие их семантическое назначение: `PetriNetType` задает тип элемента (позиция, переход, дуга); `PetriNetName` определяет уникальное имя в рамках данной модели; `PetriNetMarking`, имеющий смысл только для позиций, определяет текущую разметку в виде числа токенов и т.д. Таким образом, помимо собственно атрибутов, определяющих графическое представление, каждая фигура содержит дополнительные атрибуты, используемые расширением для анализа сетей Петри, позволяющее рассматривать данную графическую модель в виде абстрактной модели сети Петри с разметкой.

Рассмотренный подход лежит в основе программного расширения *VTMine for Visio*. Так, добавление *пользовательских атрибутов* на уровень *документа* позволяет определить такой документ как документ специального вида — *VTMine for Visio*-документ. При загрузке такого документа осуществляется активизация подсистемы программного расширения *VTMine for Visio*, загрузка соответствующих элементов пользовательского интерфейса и др. Далее мы будем обозначать эти документы как *VTMDoc*. Добавление *пользовательских атрибутов* на уровень *страницы* позволяет указать, что данная страница содержит не просто графический рисунок, но *графическую модель* (или ее часть), за взаимодействие с которой отвечает соответствующий компонент или подсистема *VTMine for Visio*. Наконец, добавление *пользовательских атрибутов* на уровень *фигуры* позволяет определить, что данная фигура является составной частью некоторой модели.

Таким образом, разработка программных расширений Visio совместно с использованием пользовательских атрибутов для ключевых компонентов объектной модели позволяет расширить его возможности до *инструмента графического моделирования*.

2. Разработка архитектуры приложения: задачи и особенности

При разработке архитектуры *VTMine for Visio* учитывались следующие требования и ограничения:

- Приложение представляет собой COM-расширение для Visio (то есть является плагином), разработанное на языке программирования платформы .NET.
- Функциональность приложения расширяется плагинами (принцип «плагины для плагина»).
- Ядро приложения содержит только базовые компоненты взаимодействия с Visio и не содержит каких-либо специфических предметным областям компонент (например, типы моделей).
- С целью рационального использования ресурсов приложение активизируется только при работе с документами *VTMine for Visio*.

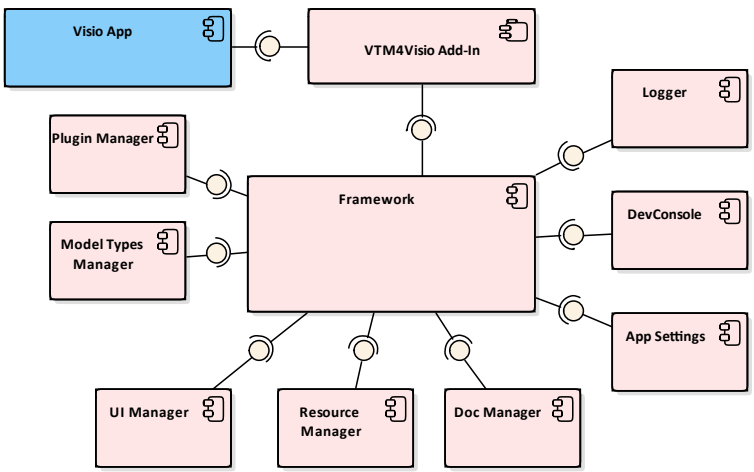


Fig. 2. *VTMine for Visio* basic components

Рис. 2. Ключевые компоненты *VTMine for Visio*

Разработанное приложение *VTMine for Visio* представляет собой Visio COM Add-In [25]. Основной язык программирования — C#. Технически, расширение является множеством сборок, оформленных в виде библиотек динамического подключения (.dll), манифестами и другими вспомогательными файлами. Основным файлом расширения является библиотека `ru.xidv.vtm4visio.dll`. Путь к директории с этой библиотекой обозначается символом `$(VtmAppDir)`.

Структурная схема компонентов ядра *VTMine for Visio* показана на рисунке 2. Далее рассматриваются назначение и особенности проектирования наиболее важных компонентов с учетом выделенных выше требований.

2.1. Расширение Visio (Add-In) и фреймворк приложения

Основным классом, связывающим приложение Visio (класс `Application`) с расширением, является класс `ThisAddIn` (на рисунке — *VTM4Visio Add-In*). Инициализация расширения производится посредством передачи управления коду расширения при возникновении события `Startup`. В обработчике события `Startup` происходит создание и инициализация главного компонента ядра расширения — объекта *Framework*.

Объект *Framework* (фреймворк) может находиться в нескольких состояниях. Изначальное состояние *Inactive* означает, что фреймворк не инициализирован. После успешной подготовки основных компонентов состояние меняется на *Initialized*, то есть инициализирован. Данное состояние подразумевает загрузку всех зависимостей компонента ядра (плагинов) в режиме минимального потребления ресурсов. При открытии документа *VTMDoc* активизируются все необходимые компоненты, распределяются ресурсы, подготавливаются необходимые элементы пользовательского интерфейса, а компонент фреймворка переходит в состояние *Active*. Наконец, после закрытия всех документов *VTMDoc* расширение переходит в состояние *Ready*, которое отличается от *Initialized* тем, что часть ресурсов являются активированными для быстрого старта при открытии следующего релевантного документа, а от состояния *Ready* — тем, что дорогостоящие ресурсы выгружены из памяти.

При начальной инициализации фреймворка (с состояния *Inactive* до состояния *Initialized*) происходит создание следующих компонентов ядра: *логгера, централизованного компонента настроек, отладочной консоли, менеджера документов, менеджера пользовательского интерфейса.* При последующей активации фреймворка (с состояния *Initialized* до состояния *Active*) создаются и инициализируются компоненты: *менеджер ресурсов, менеджер типов моделей, менеджер плагинов.*

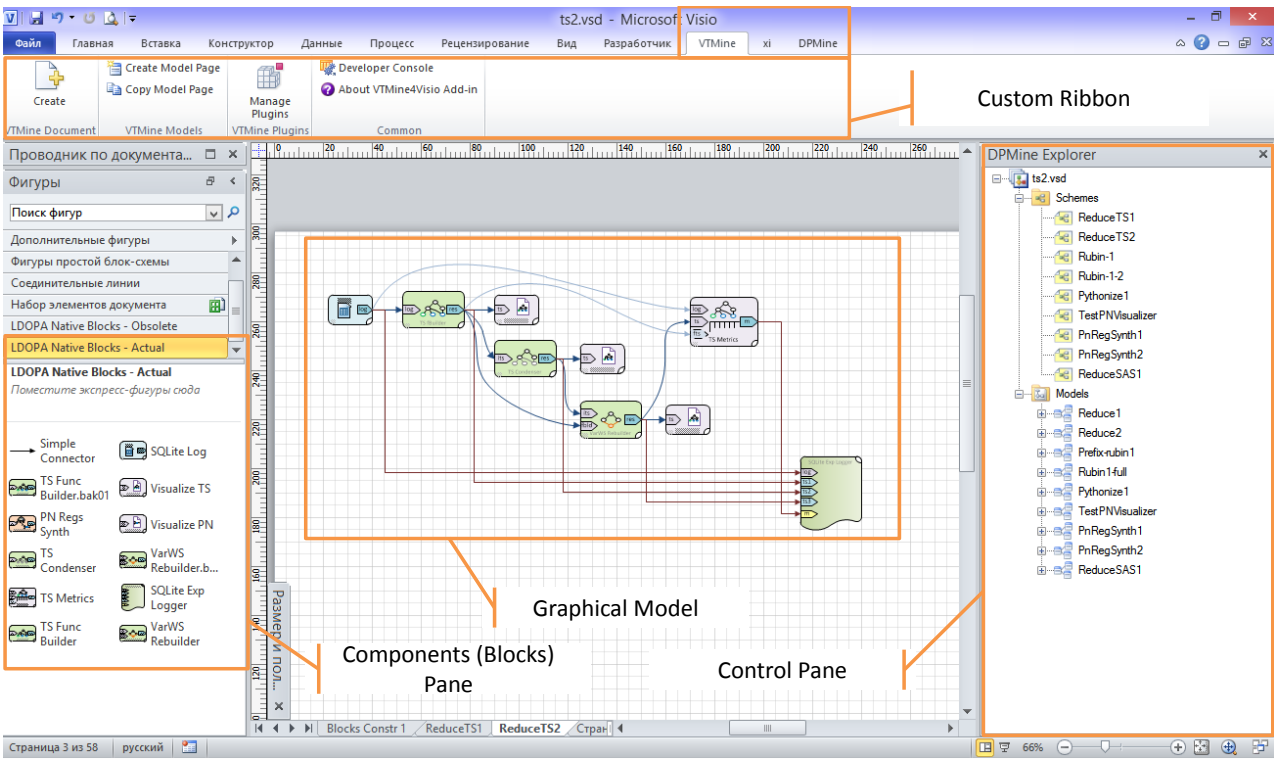


Fig. 3. Controls configured through *User Interface Manager*

Рис. 3. Элементы управления, настраиваемые через *менеджер пользовательского интерфейса*

Успешная загрузка расширения подразумевает создание расширения элемента управления «Лента»: туда добавляется новая вкладка *VTMine*, содержащая ряд управляющих компонентов для взаимодействия пользователя с надстройкой (см. рисунок 3).

Связь фреймворка с объектом Visio-приложения осуществляется путем подписки фреймворка на события приложения, среди которых события создания, открытия, закрытия документа; события переключения между окнами, смены активного окна; события добавления, удаления страницы и фигуры, смены активной страницы и изменения выделенных компонентов и др. Специальное событие `MarkerEvent` позволяет централизовано обрабатывать активацию пунктов пользовательского контекстного меню фигур и страниц, что является важным элементом взаимодействия непосредственно элементов моделей, разрабатываемых пользователем, и компонентов расширения, обрабатывающих такие модели.

В приложении используется иерархическая система распространения событий. Это означает, что события, порождаемые непосредственно объектом Visio-приложения, обрабатываются только фреймворком, который делегирует дальнейшую их обработку соответствующим компонентам, либо обрабатывает сам. Никакие другие компоненты системы не подписываются на события приложения, что позволяет управлять последовательностью их обработки и исключить проблемы, связанные с неопределенностью порядка их обработки. Последнее особенно важно с учетом возможного наличия других установленных расширений Visio.

2.1.1. Менеджер документов

Управление активными документами приложения Visio осуществляется компонентом *менджер документов* (*Document Manager*). Он является основным обработчиком следующих событий: *создание нового документа*, *открытие существующего документа* и *перед закрытием открытого документа*. При возникновении первых двух событий менеджер документов проверяет, является ли создаваемый или открываемый документ помеченным специальным образом. Для этого считывается его *пользовательский атрибут* `VTMDocument`, который определяет, что документ является документом *VTMDoc*. При первом обращении к такому типу документов осуществляет активацию фреймворка до состояния *Active*.

Создание документа *VTMDoc* имеет свои особенности. Visio предоставляет несколько способов создания нового документа. Пользователю доступны возможности создания нового документа на основе шаблона либо пустого документа. В первом случае атрибут `VTMDocument` добавляется в шаблон, поэтому все созданные на его основе документы также будут рассматриваться как *VTMDoc*. В случае создания пустого документа установка каких-либо пользовательских атрибутов невозможна. Для упрощения процедуры создания *VTMDoc*-документа без использования шаблона предусмотрена кнопка «Create» страницы *VTMine* ленты (рисунок 3).

Использование *семантических атрибутов* позволяет внедрить большое количество служебной информации, релевантной для документов *VTMDoc*, непосредственно в файл документа. Тем не менее в процессе моделирования часто создаются большие по объему ресурсы, хранение которых путем встраивания в документ не является целесообразным. Для связывания файла *VTMDoc*-документа с внешними используемыми им ресурсами разработан подход представления *VTMDoc*-документа в виде *решения* аналогично тому, как это происходит при объединении нескольких связанных файлов с исходными кодами в единый программный проект. С этой целью файл *VTMDoc*-документа (с расширением `.vsd`) рассматривается в качестве главного файла *проекта*, в дополнение к которому в директории с `.vsd`-файлом создается поддиректория с расширением .data. Например, для документа `model.vsd` его рабочая директория имеет имя `model.data`. При активизации документа `model.vsd` указанная директория доступна всем компонентам *VTMine for Visio*, включая плагины.

2.1.2. Менеджер плагинов

Одним из требований, предъявляемых к приложению *VTMine for Visio*, является возможность расширения его функционала за счет динамически загружаемых *плагинов*. Учитывая, что *VTMine for Visio* сам по себе является плагином для Visio, поставленная цель может быть достигнута двумя основными путями: 1) дополнительные модули загружаются в виде самостоятельных Visio Add-In; 2) существует единственный Visio Add-In (*VTMine for Visio*), который самостоятельно управляет загрузкой динамических расширений.

Первый подход обладает рядом недостатков, связанных с необходимостью синхронизации отдельных модулей друг с другом, отсутствием возможности контроля порядка загрузки отдельных расширений, что порождает ряд сложно решаемых проблем. Второй подход требует разработки подсистемы управления расширениями, однако позволяет централизованно ими управлять, что снимает ряд проблем и делает его предпочтительным. Именно он реализован в *VTMine for Visio*. В результате использования единственной точки загрузки динамических модулей для *VTMine for Visio* система плагинов приобретает вид, как на рисунке 4.

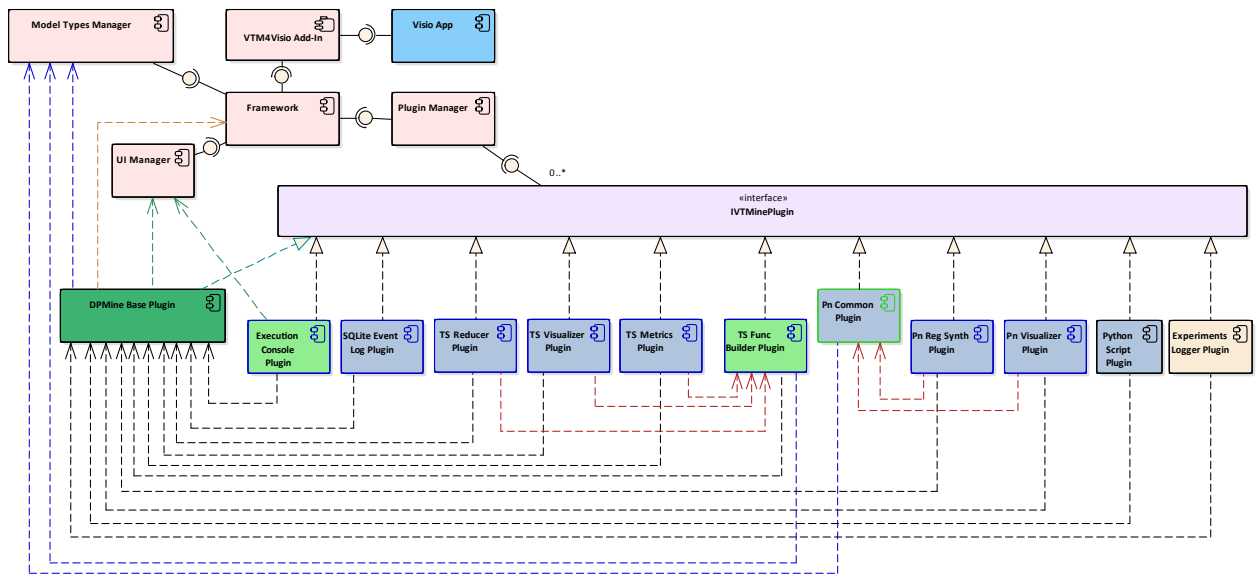


Fig. 4. Main *VTMine for Visio* plugins used in the paper

Рис. 4. Основные *VTMine for Visio*-плагины, используемые в данной работе

Плагин представляет собой специальным образом оформленный программный компонент, реализующий один или более интерфейсов, и имеющий уникальный строковый идентификатор. Основным интерфейсом, определяющим назначение компонента как VTM-плагина, является интерфейс `IVTMinePlugin`. Каждый плагин является одним из возможных *объектов расширений*, который подключается к совместимой по типу *точке расширения*. Плагины содержатся в файлах — библиотеках динамического подключения (.NET-сборках). Один такой файл может содержать несколько плагинов, что позволяет упростить распространение группы плагинов, связанных общей тематикой.

Плагины могут находиться в зависимости друг от друга, как например плагины, представленные на UML-диаграмме компонентов (рисунок 4). Большинство представленных на диаграмме плагинов имеют зависимость от плагина `DPMine Base Plugin`. Он реализует подсистему *DPMine* [12, 26] в приложении *VTMine for Visio*. Исключениями здесь являются сам плагин `DPMine Base Plugin` и плагин `Pn Common Plugin`. Это определяется тем, что в данной конфигурации приложения большинство

плагинов расширяют базовый плагин *DPMine* путем добавления новых *DPModel*-блоков либо (как в случае с плагином `Execution Console Plugin`) добавляют дополнительные компоненты пользовательского интерфейса (консоль исполнения DPM-моделей) для управления DPM-моделями.

Одним из наиболее важных компонентов ядра *VTMine for Visio* является *менеджер плагинов*. Он управляет загрузкой и инициализацией плагинов для приложения *VTMine for Visio*.

Менеджер плагинов получает управление от ядра приложения при активации последнего, то есть при переходе в состояние *Active*. Первым делом менеджер загружает конфигурационную информацию, которая хранится в специальном конфигурационном файле `plugins.json`.

Зависимость между плагинами не задается статически, а определяется динамически на этапе их загрузки. Это определяется тем фактом, что сразу несколько плагинов могут являться *объектом расширения* для одной и той же *точки расширения*. Как правило, только один из таких плагинов загружается в конкретной конфигурации, однако его идентификатор до момента загрузки неизвестен. Этот факт не позволяет построить граф зависимостей между плагинами [27] до начала их загрузки и выполнить последующую топологическую сортировку на нем для определения оптимального порядка загрузки плагинов. *Алгоритм загрузки плагинов* в упрощенном виде выглядит следующим образом.

1. Формируется список всех файлов с незагруженными плагинами — *кандидатами на загрузку*.
2. Для очередного файла из списка кандидатов загружается аннотация плагина. Аннотация включает подпрограмму, которая анализирует текущее окружение на предмет возможности удовлетворения зависимостей плагина.
3. Выполняется проверка возможности загрузки текущего плагина путем подпрограммы проверки удовлетворения зависимостей в аннотации плагина.
4. Если зависимости текущего плагина удовлетворены, он загружается и выставляется флаг наличия загруженных плагинов. Это означает, что какие-то плагины, которые не могли быть загружены на предыдущих итерациях, теперь могут быть загружены, так как загрузка текущего плагина удовлетворяет их зависимости.
5. Если есть еще плагины-кандидаты, которые не загружались в текущем цикле, выполняется переход к пункту 2.
6. Если все кандидаты проверены, остались незагруженные плагины и за текущий цикл был загружен хотя бы один плагин, выполняется переход переход к п. 1.
7. Если за текущий цикл а) все плагины были или б) ни один плагин не был загружен, — что означает наличие неразрешаемых зависимостей, — алгоритм завершается.

В худшем случае, когда последовательность плагинов для загрузки алгоритмом такова, что каждый текущий плагин зависит от следующего, сложность этого алгоритма будет квадратичной от числа плагинов. Вместе с этим в результате выполнения алгоритма плагины в текущей конфигурации загружаются в некотором определенном порядке. Если сохранить этот порядок и использовать его при следующей загрузке плагинов (при условии, что не были добавлены новые или исключены существующие плагины), то загрузка плагинов алгоритмом будет осуществлена за линейное от числа плагинов время. Сохранение оптимального порядка осуществляется в конфигурационном файле.

При загрузке некоторого плагина менеджер плагинов передает ему управление путем вызова метода `LoadPlugin`, имеющего двоичный параметр `NeedReg`. Реализация метода `LoadPlugin` индивидуальна для каждого плагина и позволяет ему взаимодействовать со всеми компонентами приложения, включая компоненты ядра и другие плагины. Взаимодействие с компонентами ядра позволяет плагину выполнять кастомизацию базовых подсистем приложения, например: регистрировать новые типы моделей (через *менеджер типов моделей*, раздел 2.1.4), добавлять перспективы и элементы управления (через *менеджер пользовательского интерфейса*, раздел 2.1.3), регистрировать

обработчики для событий документов (через *менеджер документов*, раздел 2.1.1) и т. д. Взаимодействие с другими плагинами позволяет расширять их функциональность посредством представляемого ими API. В качестве примера можно привести подсистему графического моделирования *DPMine*, которая полностью построена на плагинах и расширяется с помощью других плагинов (раздел 2.2).

Флаг `NeedReg` выставляется при самой первой загрузке этого плагина, что может быть им использовано для выполнения *первичных* операций по регистрации в системе: копирование необходимых файлов с ресурсами, инициализация конфигурационных параметров и др. При последующих загрузках зарегистрированный плагин выполняет только *динамические* операции по расширению соответствующих подсистем.

2.1.3. Менеджер пользовательского интерфейса

Взаимодействие *VTMine for Visio* с графическим пользовательским интерфейсом (ГПИ) Visio осуществляется посредством *менеджера пользовательского интерфейса*. Он включает уровень абстракции, позволяющий описывать расширение ГПИ Visio компонентами ядра и плагинами приложения, с использованием определенного набора компонентов. Среди них основными являются *перспективы*, *модификаторы ГПИ*, *элементы управления* и *фабрики элементов управления*.

К *элементам управления*, которые могут быть добавлены ядром и плагинами *VTMine for Visio*, относятся пользовательские *ленты* (ribbon), *панели управления* и *палитры компонентов* (рисунок 3). *Ленты* используются для добавления новых меню/кнопок/выпадающих списков и других простых элементов управления, посредством которых осуществляется взаимодействие пользователя и компонентов приложения. На рисунке 3 показано содержимое ленты *VTMine*, которая автоматически отображается для всех документов при инициализации основного add-in-компонента приложения. Также показано, что в меню доступна еще одна лента *DPMine*, выбор которой появляется только при активации страницы модели с типом *DPModel*. *Панели управления* — это более крупные элементы пользовательского интерфейса, позволяющие интегрировать в пользовательский интерфейс сложные диалоговые окна или масштабируемые панели со списками, иерархическими структурами и др. На рисунке показана панель *DPMine Explorer*, которая также как и лента *DPMine* активируется только для моделей типа *DPModel*. *Палитры компонентов* — это специальный тип документов, которые открываются в панели *Фигуры*, и содержат подготовленные к использованию заготовки фигур — так называемые *образцы* или *трафареты* (stencils).

Множество пользовательских элементов управления, отображаемых в каждый конкретный момент времени, определяется текущей активной *перспективой*. При переключении между страницами документа также происходит переключение между перспективами, заданными для конкретных страниц, поэтому набор видимых элементов меняется. Перспектива, назначенная некоторой странице, определяется, в первую очередь, видом модели, которая на этой странице отображается. Это позволяет отображать именно те элементы управления, которые релевантны данной конкретной модели, и скрывать остальные. Например, моделям *DPModel* по умолчанию назначается перспектива, включающая *ленту DPMine* и *панель управления DPMine Explorer*.

Менеджер пользовательского интерфейса регистрирует одну *перспективу по умолчанию*, которая отображается для всех страниц документа, для которых не назначена какая-либо другая перспектива. Эта перспектива включает ленту *VTMine*, которая содержит команду создания документа *VTMDoc*. Регистрация других перспектив осуществляется плагинами. Для этого определен специальный интерфейс *модификатора ГПИ* (`IUICustomizer`), реализовав который некоторый плагин включается в цикл управления пользовательским интерфейсом.

Таким образом, некоторый плагин, который осуществляет расширение пользовательского интерфейса, регистрирует в *менеджере пользовательского интерфейса* компонент — *модификатор ГПИ*, который реализует указанный выше интерфейс. Сам *менеджер* при активизации обращается ко всем

зарегистрированным модификаторам, запрашивая у них регистрацию новых перспектив, затем — модификацию всех зарегистрированных перспектив. При первой активизации некоторой перспективы включенные в нее элементы управления создаются путем обращения к соответствующему модификатору, после чего отображаются в приложении.

2.1.4. Менеджер типов моделей

Основное назначение *VTMine for Visio* — просмотр и редактирование графических моделей. *Документ* Visio рассматривается в качестве коллекции графических моделей, связанных общим предметом исследования. Одна модель может располагаться на одной или более *страницах* документа. В последнем случае речь идет о многостраничных моделях, например, иерархических.

VTMine for Visio не имеет фиксированного набора разновидностей моделей. Вместо этого новые типы моделей могут добавляться в систему с помощью плагинов. Управление типами моделей осуществляется компонентом ядра системы — *менеджер типов моделей*.

Разновидность (тип) модели описывается с помощью *дескриптора модели*. Дескриптором модели является объект, реализующий интерфейс `IModelDescriptor`. Такой объект предоставляет информацию об идентификаторе, наименовании типа модели, компоненте (плагине), зарегистрировавшем дескриптор. Одно из основных предназначений дескриптора модели — создавать правильным образом подготовленные страницы Visio. Запрос к дескриптору на создание модели может осуществляться со стороны компонента ядра (например, при выборе пользователем соответствующей команды) или плагина (например, при синтезе новой модели в результате работы некоторого алгоритма).

2.2. Подсистема моделирования *DPMine*

Одним из основных расширений приложения *VTMine for Visio*, реализованных в рамках настоящей работы, является подсистема поддержка моделирования экспериментов Process Mining с помощью графического языка *DPMine* [12, 26]. В настоящем разделе представляются базовые архитектурные подходы, лежащие в основе этого расширения. Демонстрируются возможности *VTMine for Visio* по интеграции с библиотекой *DPModel* [28], а также с библиотекой *LDOPA* алгоритмов и структур данных для Process Mining [29—31].

На рисунке 4 представлена компонентная диаграмма основных плагинов, относящихся к подсистеме *DPMine*. На диаграмме видно, что все плагины реализуют интерфейс `IVTMinePlugin`, посредством которого они регистрируются в *менеджере плагинов* и взаимодействуют с приложением. Представленные на рисунке плагины имеют следующее назначение (в скобках после названия плагина указан язык программирования, на котором он разработан, и библиотека, функциональность которой плагин вводит в подсистему *DPMine*).

- *DPMine Base Plugin* (C#) является *базовым плагином* подсистемы *DPMine* и определяет ее взаимодействие с приложением *VTMine for Visio*.
- *Execution Console Plugin* (C#) расширяет функциональность *базового плагина*, представляя реализацию *консоли исполнения модели*. Базовый плагин предоставляет только *точку расширения* для включения такой консоли, оставляя возможность для конкретной реализации другим плагинам.
- *SQLite Event Log Plugin* (C++/CLI, библиотека *LDOPA*) добавляет в подсистему *DPMine* определение нового блока *DPModel*, предоставляющего абстрактный интерфейс *журнала событий*, который реализован в виде журнала событий SQLite [29].
- *TS Func Builder Plugin* (C++/CLI, библиотека *LDOPA*) добавляет блок *DPModel* для синтеза систем переходов по журналу событий [32] и определение типа модели *VTMine "система переходов с частотными пометками"*. На примере этого плагина ниже приводятся особенности расширения подсистемы *DPMine*.

- *TS Reducer Plugin* (C++/CLI, библиотека *LDOPA*) добавляет блоки *DPModel* для поэтапного выполнения редукции систем переходов в соответствии с алгоритмом, представленным в [32].
- *TS Visualizer Plugin* (C++/CLI) добавляет блок *DPModel* для визуализации моделей — *систем переходов с частотными пометками*.
- *TS Metrics Plugin* (C++/CLI, библиотека *LDOPA*) добавляет блоки *DPModel* для расчета метрик систем переходов [32].
- *Pn Common Plugin* (C++/CLI) включает общие компоненты для работы с сетями Петри и добавляет определения типа соответствующей модели *VTMine*.
- *Pn Reg Synth Plugin* (C++/CLI, библиотека *LDOPA*) добавляет блок *DPModel* для выполнения синтеза сетей Петри по системам переходов модифицированной версией алгоритма регионов [33].
- *Pn Visualizer Plugin* (C++/CLI) добавляет блок *DPModel* для визуализации моделей — *сетей Петри*.
- *Experiments Logger Plugin* (C#) добавляет блок *DPModel* для конфигурируемого сбора и регистрации в базе данных расчетных значений, получаемых при выполнении эксперимента/исполнении модели *DPModel*.
- *Python Script Plugin* (C++/CLI) добавляет блок *DPModel* для выполнения произвольного Python-скрипта в контексте исполнения модели *DPModel* [31].

Все обозначенные плагины зависят от *базового плагина*, `DPMine Base Plugin`. Проверка на удовлетворение такой зависимости осуществляется процедурой `LoadPlugin` при загрузке очередного плагина. Если *базовый плагин* еще не загружен к моменту загрузки очередного плагина, то загрузка последнего откладывается. Помимо зависимости от *базового плагина* некоторые из отображенных на схеме плагинов имеют и другие зависимости. Так, плагины *TS Reducer Plugin*, *TS Visualizer Plugin* и *TS Metrics Plugin* зависят от плагина *TS Func Builder Plugin*, а плагины *Pn Reg Synth Plugin* и *Pn Visualizer Plugin* — от плагина *Pn Common Plugin* (рисунок 4). Характер зависимости плагинов друг от друга может быть разным. В одних случаях зависимые плагины расширяют функционал плагина-зависимости; большинство плагинов в примере выше расширяют функционал базового плагина. В других случаях зависимые плагины используют некоторые компоненты плагина-зависимости; так, *Pn Common Plugin* содержит базовые определения для моделей на основе сетей Петри, и зависящие от него плагины используют их.

2.2.1. Базовый плагин *DPMine* и его расширение

Принцип расширения одного плагина другими рассматривается на примере *базового плагина DPMine* (`DPMine Base Plugin`). Структура основных компонентов, входящих в состав этого плагина, представлена на диаграмме классов (рисунок 5). На диаграмме помимо компонентов базового плагина представлены некоторые связанные с ним компоненты ядра, другие плагины и библиотеки. Диаграмма включает только часть внутренних и внешних компонентов, которые позволяют проиллюстрировать механизм организации зависимостей компонентов и расширения их друг другом. Внешние по отношению к базовому плагину компоненты заключены в пунктирные прямоугольники и представляют следующие *ограниченные домены* (boundaries).

- `Visio` — содержит основные компоненты приложения Visio, такие как *документ* и *страница*.
- `VTM4Visio Core` — компоненты ядра, на которые влияют загружаемые плагины.
- `Execution Console Plugin` — плагин *консоли исполнения модели*. На диаграмме представлен только своим основным классом `ExecConsolePlugin`.
- `DPModel Library` — библиотека компонентов *DPModel* [28], включающая реализацию языка моделирования *DPMine* и некоторых базовых блоков [26]. Библиотека написана на языке C# и является независимой от инструмента *VTMine for Visio*. Плагины подсистемы *DPModel* инструмента используют ее как внешнюю зависимость.

- `LDOPA Library` — библиотека алгоритмов и структур данных для Process Mining [29, 31]. Библиотека написана на языке C++ и является внешней зависимостью для приложения.
- `TS Func Builder Plugin` — плагин, расширяющий функциональность ядра путем добавления нового типа моделей “системы переходов с частотами” (компонент `FreqTsMD`); также расширяет функциональность *базового плагина* путем регистрации нового типа *DPModel*-блока (`TsFuncBuilderBlock`), который выполняет синтез системы переходов с частотами *префиксного дерева* по журналу событий. Плагин осуществляет интеграцию компонента синтеза библиотеки *LDOPA* в систему моделирования экспериментов *DPMine*.

Базовый плагин DPMine включает следующие основные компоненты.

- *Проводник моделей* DPMine, представленный классом `DpmExplorer`, реализует интерфейс ядра `IUICustomizer`, осуществляет добавление в перспективу моделей *DPModel* управляющий элемент — *панель управления* с иерархическим проводником по моделям *DPMine*, зарегистрированным в текущем документе (см. рисунок 3, справа).
- *Консоль исполнения модели*, представленная интерфейсом `IExecConsole`, реализует расширение пользовательского интерфейса приложения путем добавления *панели управления*, отображающую информацию об исполнении модели *DPModel*. В отличие от *проводника моделей консоль исполнения* не является компонентом, встроенным в базовый плагин. Вместо этого другие плагины могут представлять свою специфическую реализацию консоли и подключать ее к базовому плагину. В данном случае в роли такого плагина выступает класс `ExecConsolePlugin`.
- *Дескриптор блока DPModel*, представленный интерфейсом `IBlockDescriptor`, используется для регистрации в подсистеме *DPMine* типа блока, который может быть инстанцирован в модели *DPModel*. Класс `BlockDescriptor` реализует интерфейс `IBlockDescriptor` и представляет базовый функционал для дескрипторов большинства блоков. Примерами блоков и соответствующих им дескрипторов, регистрируемых непосредственно *базовым плагином*, являются `DPMineBaseBlock`/`DPMineBaseBD` и `DPMineScheme`/`DPMineSchemeBD`. Они представляют *базовый блок* и *схемный блок* модели *DPModel* соответственно [12, 26].
- *Дескриптор типа модели* DPModel представлен классом `DPMineMD`, который реализует интерфейс `IModelDescriptor`. Дескриптор моделей *DPModel* регистрируется в *VTMine for Visio* с помощью *менеджера типов моделей*.
- *Модель* DPModel представлена классом `DPMineModel`. Это невизуальная объектная модель, в которую преобразуется графическая модель эксперимента, составленная в Visio с помощью блоков и коннекторов. Класс `DPMineModel` расширяет функционал класса `ModelImpl`, определенного в библиотеке *DPModel*, а также реализует интерфейс `IVTModel` моделей уровня приложения *VTMine for Visio*.
- *Менеджер моделей* DPModel представлен классом `ModelManager`, используется для регистрации и исполнения моделей *DPModel*, а также координирует взаимодействие этих моделей с окружением *VTMine for Visio*. Модели *DPModel* группируются по документам Visio с помощью вспомогательного компонента DPModel-*профиль документа*, который представлен классом `DocDpmProfile`.

Плагин `TS Func Builder Plugin` содержит компоненты для синтеза систем переходов по журналам событий. Алгоритм синтеза реализован как часть библиотеки *LDOPA* и представлен на диаграмме классами `TsBuilder` и `PrefixStateFunc`[2].

Вместе с этим плагин также расширяет систему *типов блоков* DPModel с помощью *дескриптора типа блоков* `TsFuncBuilderBD`, наследующего у класса `BlockDescriptor` по аналогии с дескрипторами `DPMineBaseBD` и `DPMineSchemeBD`. Одна из основных задач, которые возлагаются на *дескрипторы блоков* DPModel, — это конструирование блоков невизуальной (объектной) *модели* DPModel по их

[2]Используется для выведения состояния системы переходов с помощью префиксов [32, 34].

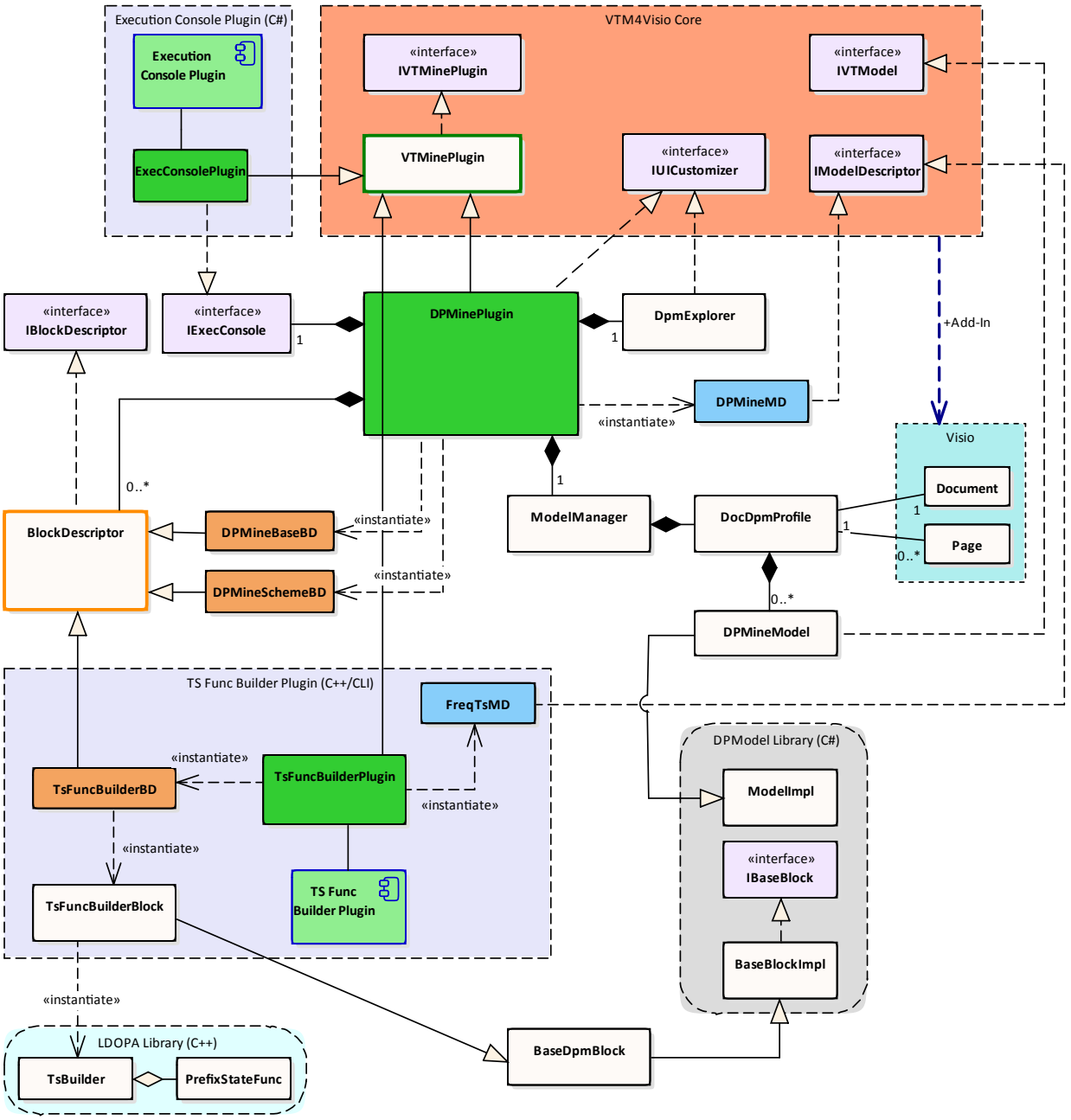


Fig. 5. Components of the *DPMine* Base Plugin and their interaction with libraries and other plugins (external dependencies are represented by dashed rectangles)

Рис. 5. Компоненты базового плагина *DPMine* и их взаимодействие с библиотеками и другими плагинами (внешние зависимости представлены пунктирными прямоугольниками с прямыми или скругленными углами)

графическому представлению, которое задается фигурой Visio (классом `Shape`). Сконструированный таким образом блок (для плагина синтеза систем переходов задается классом `TsFuncBuilderBlock`) впоследствии добавляется в модель *DPModel* и соединяется коннекторами с другими блоками в соответствии с тем, как их визуальные представления соединены коннекторами на уровне графической модели.

3. Примеры моделей экспериментов Process Mining

В данном разделе приводятся примеры моделей экспериментов Process Mining, отражающие следующие операции: работу с журналом событий; синтез моделей процессов по журналу событий и по другим моделям процессов; визуализацию моделей процессов; расчет характеристик моделей (метрик); сохранение произвольной информации в структурированном виде (БД) для последующей статистической обработки.

Основные принципы организации *моделей DPModel* рассматриваются в [12, 26]. *Графическая модель DPModel* транслируется в *объектную модель* и затем исполняется. *Графическая модель* состоит из одной или нескольких схем, каждая схема представляется в виде одной страницы Visio. На рисунке 3 представлен пример схемы с именем `ReduceTS2`, расположенной на одноименной странице (список страниц отображается в нижней части экранной формы), которая является *главной схемой* для модели `Reduce2` (список моделей отображен в *проводнике моделей* в правой части окна).

Фигура Visio (объект класса `Shape`), представляющая блок, помечается атрибутом `User.DPMine_Type = ''block''`. Атрибут `User.DPMine_Block_Type` определяет конкретное значение идентификатора типа блока, регистрируемого в системе с помощью соответствующего дескриптора типа блока.

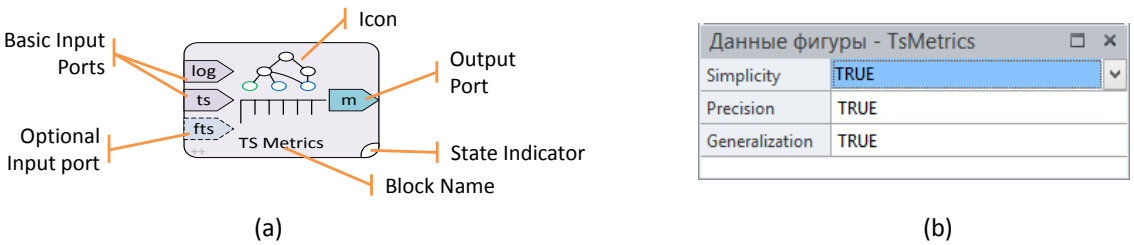


Fig. 6. Graphic notation (a) and the parameter panel (b) of the graphic block for *calculation of transition system metrics*

Рис. 6. Условное графическое обозначение (a) и панель параметров (b) графического блока *расчета метрик систем переходов*

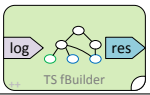
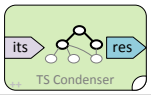
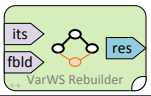
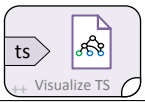
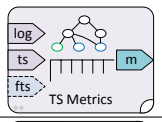
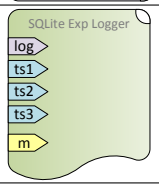
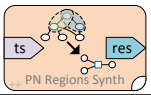
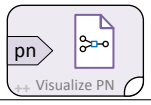
Графическое представление блока может быть любым, однако система конвертации ожидает наличие в фигуре блока вложенных фигур, атрибутированных специальным образом, которые представляют специальные элементы блока. На рисунке 6*a* представлен блок *расчета метрик систем переходов*, добавляемый в систему плагином *TS Metrics Plugin*. Фигура блока содержит ряд вложенных фигур, обозначающих *входные* и *выходные порты* (аннотированные атрибутами `User.DPMine_Type = ''port''` и `DPMine_Pore_Dir = ''in''` и `''out''` соответственно), *индикатор выполнения* (аннотирован атрибутом `User.DPMine_Type = ''execindicator''`) и текстовые, и графические обозначения блока. Блок содержит ряд параметров, доступных для редактирования пользователем через панель инструментов *«Данные фигуры»* (рисунок 6*b*). Параметры задаются с помощью переключателей `TRUE`/`FALSE`, что обозначает соответственно выполнение или невыполнение расчета каждой из трех метрик. Эти параметры ассоциируются с конкретной фигурой блока с помощью атрибутов `Prop.calc_simplicity`, `Prop.calc_precision` и `Prop.calc_general`.

Перечень блоков, используемых в разработанных моделях, приведен в таблице 1.

3.1. Модель «Синтез системы переходов по журналу событий»

Простейшая *DPModel*-модель синтеза системы переходов по журналу событий представлена на рисунке 7. Модель представлена одной схемой, которая состоит из трех последовательно соединенных блоков: *блока SQLite-журнала событий*, *блока параметрического синтеза систем переходов* и *блока визуализации помеченной системы переходов*.

Table 1. *DPModel* blocks description **Таблица 1.** Блоки *DPModel*

Блок	Описание
	Доступ к журналу событий в формате *SQLite EventLog* [29]. Плагин *SQLite Event Log Plugin*
	Синтез системы переходов по журналу событий [32]. Плагин *TS Func Builder Plugin*
	Построение конденсированной системы переходов (шаг 2 алгоритма редукции; [32]). Плагин *TS Reducer Plugin*
	Восстановление конденсированной системы переходов до редуцированной (шаг 3 алгоритма редукции; [32]). Плагин *TS Reducer Plugin*
	Визуализатор помеченной системы переходов. Плагин *TS Visualizer Plugin*
	Расчет метрик системы переходов [32]. Плагин *TS Metrics Plugin*
	Логирование результатов экспериментов в БД SQLite. Плагин *Experiments Logger Plugin*
	Синтез сети Петри по системе переходов методом регионов [33]. Плагин *Pn Reg Synth Plugin*
	Визуализатор сетей Петри. Плагин *Pn Visualizer Plugin*

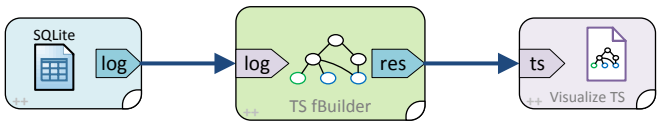


Fig. 7. Model of the synthesis of transition systems based on event logs, and their visualization

Рис. 7. Модель синтеза системы переходов по журналу событий и визуализация модели

Блок журнала событий содержит параметры: *имя файла журнала* (*File Name*) и *строка с конфигурационным SQL-выражением* (*Get Config Query*; [29]). *Блок синтеза систем переходов* содержит параметры: *размер окна* (*Wnd Size*) и *функция, используемая для выведения состояния* (*State Function*). Значение –1 в качестве размера окна обозначает, что оно не ограничено, а `Prefix` в качестве *функции выведения состоянии* задает использование префиксов [32, 34]. Эта совокупность параметров конфигурирует *блок синтеза системы переходов* на построение *префиксного дерева*. Такая конфигурация используется, например, при визуализации журнала событий.

При исполнении модели, *блок журнала событий* открывает файл БД журнала и подготавливает запросы в соответствии с конфигурацией, извлекаемой из журнала конфигурационным запро-

сом [29]. *Блок синтеза систем переходов*, подключенный при помощи коннектора своим входным портом к выходному порту *блока журнала событий* синтезирует систему переходов [32], которая подается на вход *блока визуализации системы переходов*. Этот последний блок создает новую страницу-модель уровня приложения *VTMine for Visio* и отрисовывает на ней синтезированную систему переходов с помощью аннотированных фигур Visio.

Сама страница модели также аннотируется с целью добавления справочной информации, содержащей временную метку создания модели, источник данных для нее, параметры алгоритма синтеза и др. Пример синтезированного с помощью такой модели префиксного дерева и аннотации к нему приведен на рисунке 8.

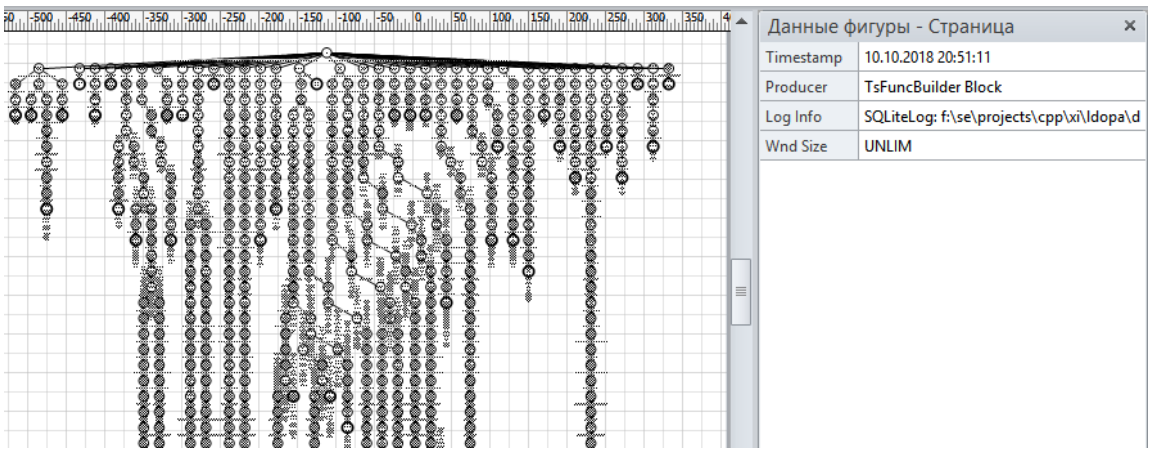


Fig. 8. A prefix tree model and its annotation with information about the source of the model

Рис. 8. Модель «префиксное дерево» и ее аннотация с информацией об источнике модели

3.2. Модель «Синтез и редукция системы переходов с расчетом метрик»

Более сложная версия модели эксперимента, включающая синтез префиксного дерева по журналу событий с последующей редукцией и расчетом метрик редуцированной системы [32], представлена на рисунке 3. Модель синтеза с редукцией является расширением модели синтеза (рисунок 7) и включает следующие новые блоки: *блок построения конденсированной системы*, *блок восстановления конденсированной системы переходов до редуцированной*, *блок расчета метрик системы переходов* и *блок логирования результатов эксперимента в БД SQLite* (см. таблицу 1 для расшифровки графических обозначений блоков).

Данная модель включает в себя полный цикл алгоритма редукции системы переходов, описанный в [32]. Шаги 1, 2 и 3 алгоритма выполняются отдельными блоками, позволяющими изменять отдельные параметры синтеза, конденсации и восстановления, а также — строить графические представления промежуточных моделей. Для этого в модели присутствует три *блока визуализации систем переходов*, подключенных коннекторами к соответствующим выходным портам блоков синтеза, конденсации и восстановления[3].

При исполнении модели подготавливается журнал событий и осуществляется синтез префиксного дерева. После этого становится активным *блок конденсации*, который на основе префиксного дерева строит конденсированное дерево. Оба этих дерева являются обязательными входными параметрами для *блока восстановления*, поэтому его входные порты соединены с выходными портами соответственно *блока синтеза системы переходов* и *блока конденсации*. Результатом работы *блока вос-*

[3]Разные цвета, форма и толщина линий коннекторов, изображенных на схеме модели, несут одинаковую семантическую нагрузку.

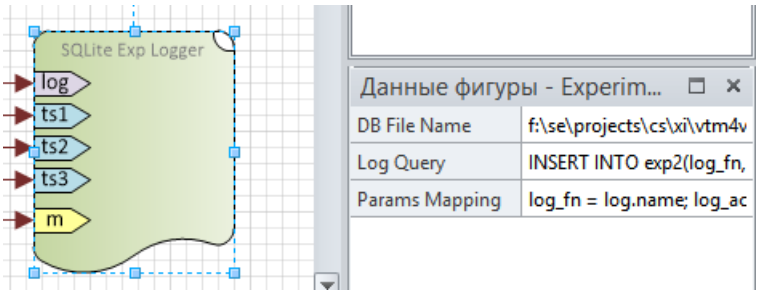


Fig. 9. *Experiment logging block* parameters

Рис. 9. Параметры *блока логирования результатов эксперимента*

становления является результирующая *редуцированная система переходов*, которая визуализируется вместе с двумя другими промежуточными деревьями.

После проведения полного цикла синтеза редуцированной системы переходов становится возможным рассчитать ее метрики, что осуществляется *блоком расчета метрик*. Он содержит три входных порта, которые соединяются с выходными портами *блока журнала событий* (для получения некоторой статистической информации об исходном журнале), *блоком синтеза системы переходов* (для получения характеристик полной системы/префиксного дерева) и *блоком восстановления* — метрики рассчитываются для редуцированной системы переходов, порождаемой именно этим блоком.

Самым последним блоком, который исполняется в этой модели эксперимента, является *блок логирования результатов эксперимента*. Этот блок может иметь произвольное число входных портов, которые должны иметь различимые имена. *Блок логирования* используется для извлечения контрольных данных из подключенных к нему блоков и сохранения их в базу данных, накапливающую результаты экспериментов для их последующего анализа. Этот блок имеет три параметра, настраиваемых пользователем (рисунок 9). Параметр *DB File Name* задает имя файла БД, в который осуществляется запись результатов. Параметр *Log Query* определяет SQL-выражение, осуществляющее добавление одной записи со значениями параметра экспериментов в БД (в одну или несколько таблиц). Параметр *Params Mapping* содержит конфигурационную строку, осуществляющую связь между именами портов и объектами, передаваемыми через них, с параметрами SQL-запроса.

Для рассматриваемой модели SQL-запрос на добавление записи в БД выглядит следующим образом:

```
INSERT INTO exp2(log_fn, log_acts_num, log_traces_num, .....)
VALUES (@log_fn, @log_acts_num, @log_traces_num, .....)
```

Этот запрос осуществляет вставку одной записи в таблицу `exp2` значений атрибутов, заданных именами: `log_fn`, `log_acts_num` и т. д. Значения связываются с запросом посредством соответствующих параметров: `@log_fn`, `@log_acts_num`. Эти параметры, в свою очередь, связываются с внешним окружением *блока логирования* с помощью его параметра *Params Mapping*, принимающего в данном примере следующее значение:

```
log_fn = log.name; log_acts_num = log.actsn; log_traces_num = log.tracesn; .....
```

Params Mapping представляет собой строку пар «параметр = значение», разделенных точкой с запятой. Первый элемент пары представляет собой одноименный, начинающийся со знака `@`, параметр приведенного выше SQL-выражения, второй — выражение, описывающее связь SQL-параметра с входным портом. Например, запись `log_fn = log.name` означает, что значение SQL-параметра `@log_fn` принимается с порта с именем `log`.

Table 2. Database fragment with logged experiment results

Таблица 2. Фрагмент БД с логированными результатами экспериментов

time_st	e...	log_fn	log_acts_num	log_traces_num	log_events_num	ts1_wndsize	ts1_sts_num	ts1_trs_
Click here to define a filter								
2018-10-07 22:59:19.000	(null)	SQLiteLog: F:\se\projects\cpp\xi\ldopa\dev\root\0.1\tests\gtest\work_files\logs\log05.sq3	7	8	41	6	16	
2018-10-07 23:03:51.000	(null)	SQLiteLog: F:\se\projects\cpp\xi\ldopa\dev\root\0.1\tests\gtest\work_files\logs\log05.sq3	7	8	41	6	16	
2018-10-26 13:55:22.000	(null)	SQLiteLog: F:\se\projects\cpp\xi\ldopa\dev\root\0.1\tests\gtest\work_files\logs\log05.sq3	7	8	41	6	16	
2019-04-28 18:37:48.000	(null)	SQLiteLog: F:\research\topics\SoftwarePM\SRA\logs\db\set1\db\all-shorts.sq3	143	243	2444	160	1337	

Входные порты *блога логгера* могут быть подключены к выходным портам тех блоков, которые предоставляют *мультиплексированный ресурс*. *Мультиплексированный* или *композитный ресурс* представляется с помощью объекта, реализующего более одного интерфейса типа ресурсов. Например, *блок синтеза системы переходов* формирует на своем выходном порту *мультиплексированный ресурс*, включающий следующие объекты: *систему переходов* (через нативный тип C++), *журнал событий* (через нативный абстрактный тип C++), *функцию выведения состояния* (через нативный абстрактный тип C++), *список тэгов* (через виртуальное свойство C#), *получатель некоторого свойства* (через виртуальный метод C#). *Выходной порт*, на котором формируется мультиплексированный ресурс, также называется *мультиплексированным*.

Нативные объекты C++, доступные через указанный мультиплексированный порт, могут быть использованы другими блоками, подключенными коннекторами к этому порту, если они умеют работать с нативными объектами (то есть такие блоки сами должны быть реализованы на языке C++/CLI). Так, *блок конденсации*, подключенный к выходному мультиплексированному порту *блока синтеза*, извлекает синтезированную им систему переходов в виде нативного объекта.

Список тегов представляется в виде C#-коллекции *троек строк*. В частном случае каждая такая строка описывает некоторую характеристику некоторого объекта, который полагается вместе со списком тегов на выходном мультиплексированном порте. В примере такой список будет характеризовать *синтезированную систему переходов* следующими атрибутами: временной отметкой создания модели, журнала события, на основе которого эта система синтезирована и т. д. Нетрудно заметить, что именно эта информация выводится *блоком визуализации системы переходов* в виде свойств страницы, на которой данная модель визуализируется (рисунок 8).

Получатель свойства декларируется интерфейсом уровня ядра с именем `IGetProperty` и описывает единственный метод с C#-сигнатурой `Object GetProperty(Object key)`. Этот метод получает произвольный параметр (например, строковый), и возвращает связанное с ним значение. Именно этот метод используется *блоком логирования* для извлечения необходимой ему информации. В рассматриваемом выше примере — выражении `log_fn = log.name` — SQL-параметр `@log_fn` принимает свое значение с *мультиплексированного порта* `log`, на котором формируется ресурс, реализующий интерфейс `IGetProperty`. Вторая часть записи `log.name` (а именно `name`) задает строковый параметр `key`, который передается в метод `GetProperty()`. Полученное значение приводится к совместимому с БД SQLite типу и привязывается к параметру `@log_fn`, после чего попадает в БД в виде зафиксированного значения эксперимента.

Таким способом осуществляется фиксация результатов отдельных экспериментов: характеристик журнала событий, параметров алгоритмов синтеза и редукции и рассчитанных метрик результирующей модели. Фрагмент БД с результатами экспериментов, зафиксированными с помощью *блока логирования*, представлен в таблице 2.

3.3. Модель «Синтез сети Петри по журналу событий методом регионов через промежуточную систему переходов»

На рисунке 10 представлена модель эксперимента, которая осуществляет синтез целевой модели — *сети Петри* алгоритмом регионов [33] через промежуточную систему переходов, построенную по журналам событий. Эта модель является расширением модели на рисунке 7: к выходному порту *блока синтеза системы переходов* помимо *блока визуализации* подключается *блок синтеза сети Петри*.

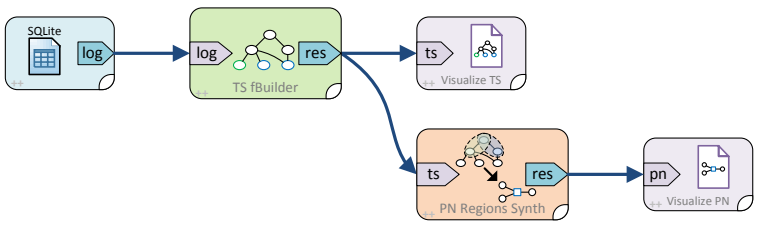


Fig. 10. Model for Petri nets synthesis based on event logs by the algorithms of regions using an intermediate transition system

Рис. 10. Модель синтеза сети Петри по журналу событий алгоритмом регионов через промежуточную систему переходов

Блок визуализации сетей Петри работает аналогично *блоку визуализации систем переходов*, создавая новую страницу-модель и размещая на ней графическое представление синтезированной сети Петри в виде набора связанных фигур. Пример сети Петри, синтезированной с помощью такой модели, представлен на рисунке 11.

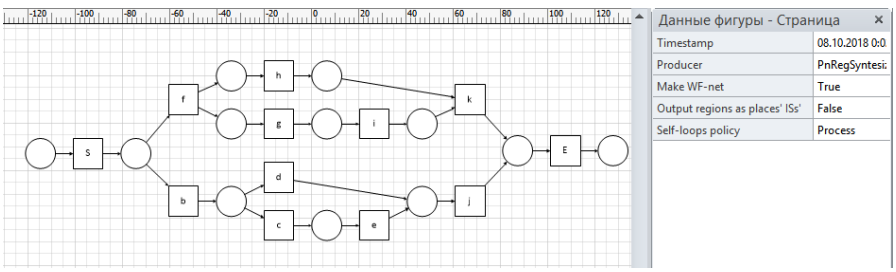


Fig. 11. A Petri net synthesized by the algorithm of regions using a reduced transition system

Рис. 11. Сеть Петри, синтезированная алгоритмом регионов по редуцированной системе переходов

Заключение

В работе рассмотрены принципы разработки графического инструмента моделирования для предметной области извлечения и анализа процессов. Предложен подход по расширению существующего инструмента бизнес-аналитики MS Visio, основанный на семантических атрибутах компонентов Visio. Предложенный подход реализован в виде приложения — расширения (add-in) для MS Visio и набора сопутствующих плагинов, ознакомиться с которыми можно на сайте проекта https://prj.xiart.ru/projects/vtmine4visio. Работа инструмента продемонстрирована на наборе схем экспериментов по синтезу и анализу моделей процессов. В предложенных примерах рассматриваются возможности автоматического построения графических моделей процессов, а также логирования информации о проведенных экспериментах в виде БД для последующей обработки.

Разработанный инструмент использует высокопроизводительную библиотеку *LDOPA* алгоритмов Process Mining, что позволяет расширять его функциональность за счет включения новых

блоков алгоритмов, реализованных в этой библиотеке. Среди направлений будущей работы можно отметить расширение числа поддерживаемых алгоритмов и типов моделей.

References

[1] W. M. P. Van Der Aalst, *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011, pp. I–XVI, 1–352, ISBN: 978-3-642-19344-6.

[2] M. A., K. A., S. S., and W. M. P. Van Der Aalst, "Using process mining for the analysis of an e-trade system: A case study", *Business Informatics*, vol. 29, no. 3, pp. 15–27, 2014.

[3] V. Rubin, L. I., and W. M. P. Van Der Aalst, "Agile Development with Software Process Mining", in *In proceedings: ICSSP 2014, Nanjing, Jiangsu, China*, ACM, 2014, pp. 70–74.

[4] V. Rubin, A. A. Mitsyuk, I. A. Lomazova, and W. M. P. 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*, NY: ACM, 2014.

[5] R. S. Mans, M. H. Schonenberg, M. Song, W. M. P. Van Der Aalst, and P. J. M. Bakker, "Application of Process Mining in Healthcare – A Case Study in a Dutch Hospital", in *Biomedical Engineering Systems and Technologies*, A. Fred, J. Filipe, and H. Gamboa, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 425–438, ISBN: 978-3-540-92219-3.

[6] R. S. Mans, W. M. P. Van Der Aalst, R. J. B. Vanwersch, and A. J. Moleman, "Process Mining in Healthcare: Data Challenges When Answering Frequently Posed Questions", in *ProHealth/KR4HC*, R. Lenz, S. Miksch, M. Peleg, M. Reichert, D. Riaño, and A. ten Teije, Eds., ser. Lecture Notes in Computer Science, vol. 7738, Springer, 2012, pp. 140–153, ISBN: 978-3-642-36437-2. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-36438-9_10.

[7] *Visio Website*. [Online]. Available: https://products.office.com/en-us/visio/flowchart-software.

[8] M. Kebede and M. Dumas, "Kebede, M. and Dumas, M", *University of Tartu*, 2015.

[9] U. Celik and E. Akçetin, "Process mining tools comparison", *Online Academic Journal of Information Technology*, vol. 9, pp. 97–104, 2018.

[10] H. Verbeek, J. Buijs, B. Van Dongen, and W. Van Der Aalst, "ProM 6: The Process Mining Toolkit", in *Proc. of BPM Demonstration Track*, ser. CEUR Workshop Proceedings, vol. 615, 2010, pp. 34–39.

[11] R. Mans, W. M. P. Van Der Aalst, and H. Verbeek, "Supporting Process Mining Workflows with RapidProM", in *Proceedings of the BPM Demo Sessions 2014*, vol. 56, 2014.

[12] S. Shershakov, "DPMine/P: modeling and process mining language and ProM plug-ins", in *Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia*, ACM New York, NY, USA, 2013, ISBN: 978-1-4503-2641-4. DOI: 10.1145/2556610.2556622. [Online]. Available: http://dl.acm.org/citation.cfm?id=2556622&CFID=415147702&CFTOKEN=35395117.

[13] *Fluxicon*. [Online]. Available: http://fluxicon.com/disco.

[14] *Celonis*. [Online]. Available: www.celonis.com.

[15] *Minit*. [Online]. Available: www.minit.io.

[16] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev, "Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers", *IEICE Transactions on Information and Systems*, vol. E80-D, no. 3, pp. 315–325, 1997.

[17] J. Cortadella, M. Kishinevsky, A. Kondratyev, and L. Lavagno, "Introduction to asynchronous circuit design: specification and synthesis (tutorial)", in *Proceedings of 6th. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems, Eilat (Israel)*, 2000.

[18] M. Solé and J. Carmona, "Rbminer: A Tool for Discovering Petri Nets from Transition Systems", in *Automated Technology for Verification and Analysis*, Springer Berlin Heidelberg, 2010, pp. 396–402, ISBN: 978-3-642-15643-4.

[19] J. Carmona, J. Cortadella, and M. Kishinevsky, "Genet: A Tool for the Synthesis and Mining of Petri Nets", in *2009 Ninth International Conference on Application of Concurrency to System Design*, 2009, pp. 181–185.

[20] J. Carmona Vargas and M. Solé, "PMLAB: An Scripting Environment for Process Mining", in *Proceedings of the BPM Demo Sessions'14*, 2014, pp. 16–16.

[21] A. Berti, S. J. Van Zelst, and W. M. P. Van Der Aalst, "Process Mining for Python (PM4Py): Bridging the Gap Between Process-and Data Science", Tech. Rep., 2019.

[22] G. Janssenswillen and B. Depaire, "bupaR: Business Process Analysis in R", in *BPM*, 2017.

[23] E. W. Dijkstra, "Cooperating Sequential Processes", in *TR EWD-123*. 1965.

[24] W. Reisig, *Understanding Petri Nets, Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2010. DOI: 10.1007/978-3-642-33278-4.

[25] *Visio documentation*. [Online]. Available: https://docs.microsoft.com/en-us/office/dev/add-ins/visio/.

[26] S. A. Shershakov, "DPMine graphical language for automation of experiments in process mining", *Automatic Control and Computer Sciences*, vol. 50, no. 7, pp. 477–485, 2016, ISSN: 1558-108X. DOI: 10.3103/S014641161607018X. [Online]. Available: http://dx.doi.org/10.3103/S014641161607018X.

[27] P. Kim, O. Bulanov, and S. Shershakov, "Component-based VTMine/C Framework: Not Only Modelling", in *Proceedings of the 8th Spring/Summer Young Researchers' Colloquium on Software Engineering, SYRCoSE*, ISP RAS, 2014, pp. 102–107. [Online]. Available: http://syrcose.ispras.ru/2014/files/SYRCoSE2014_Proceedings.pdf.

[28] *DPModel Official Website*. [Online]. Available: https://prj.xiart.ru/projects/dpmodel.

[29] S. Shershakov, "Multi-Perspective Process Mining with Embedding Cofigurations into DB-based Event Logs", in *Communications in Computer and Information Science*, In Press, Springer, 2020.

[30] *LDOPA Official Website*. [Online]. Available: https://prj.xiart.ru/projects/ldopa.

[31] S. Shershakov, "Enhancing Efficiency of Process Mining Algorithms with a Tailored Library: Design Principles and Performance Assessment", National Research University Higher School of Economics, Tech. Rep., 2018. DOI: 10.13140/RG.2.2.18320.46084. [Online]. Available: https://www.researchgate.net/publication/332869308_Enhancing_Efficiency_of_Process_Mining_Algorithms_with_a_Tailored_Library_Design_Principles_and_Performance_Assessment_Technical_Report.

[32] S. A. Shershakov, A. A. Kalenkova, and I. A. Lomazova, "Transition Systems Reduction: Balancing Between Precision and Simplicity", in *Transactions on Petri Nets and Other Models of Concurrency XII*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, pp. 119–139, ISBN: 978-3-662-55862-1. DOI: 10.1007/978-3-662-55862-1_6. [Online]. Available: https://doi.org/10.1007/978-3-662-55862-1_6.

[33] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev, "Deriving Petri Nets from Finite Transition Systems", *IEEE Trans. Comput.*, vol. 47, no. 8, pp. 859–882, 1998, ISSN: 0018-9340. DOI: 10.1109/12.707587. [Online]. Available: http://dx.doi.org/10.1109/12.707587.

[34] W. M. P. Van Der Aalst, V. Rubin, H. M. W. Verbeek, B. F. Dongen, E. Kindler, and C. W. Günther, "Process mining: a two-step approach to balance between underfitting and overfitting", *Software & Systems Modeling*, vol. 9, no. 1, pp. 87–111, 2010, ISSN: 1619-1374. [Online]. Available: http://dx.doi.org/10.1007/s10270-008-0106-z.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 27, NO. 2, 2020
journal homepage: www.mais-journal.ru

ALGORITHMS

Research and Development of an Algorithm for the Response Time Estimation in Multiprocessor Systems Under the Interval Uncertainty of the Tasks Execution Times

M. G. Gonopolskiy[1], A. B. Glonina[1]

DOI: 10.18255/1818-1015-2020-2-218-233

[1]Lomonosov Moscow State University, 1 Leninskie Gory, Moscow 119991, Russia.

MSC2020: 68M20
Research article
Full text in Russian

Received February 11, 2020
After revision March 9, 2020
Accepted March 11, 2020

The paper presents an algorithm for the worst case response time (WCRT) estimation for multiprocessor systems with fixed-priority preemptive schedulers and the interval uncertainty of tasks execution times. Each task has a unique priority within its processor, a period, an execution time interval [BCET, WCET] and can have data dependency on other tasks. If a decrease in the execution time of the task A can lead to an increase in the response time of the another task B, then task A is called an anomalous task for task B. According to the chosen approach, in order to estimate a task's WCRT, two steps should be performed. The first one is to construct a set of anomalous tasks using the proposed algorithm for the given task. The paper provides the algorithm and the proof of its correctness. The second one is to find the WCRT estimation using a genetic algorithm. The proposed approach has been implemented software as a program in Python3. A set of experiments have been carried out in order to compare the proposed method in terms of precision and speed with two well-known WCRT estimating methods: the method that does not take into account interval uncertainty (assuming that the execution time of a given task is equal to WCET) and the brute force method. The results of the experiments have shown that, in contrast to the brute force method, the proposed method is applicable to the analysis of the real scale computing systems and also allows to achieve greater precision than the method that does not take into account interval uncertainty.

Keywords: genetic algorithm; multiprocessor systems; worst-case response times; response time analysis; scheduling anomalies; scheduling.

INFORMATION ABOUT THE AUTHORS

Mark G. Gonopolskiy correspondence author	orcid.org/0000-0002-0670-8603. E-mail: gmarkmgw@yandex.ru student.
Alevtina B. Glonina correspondence author	orcid.org/0000-0001-8716-4128. E-mail: alevtina@lvk.cs.msu.su programmer.

Funding: This work was supported by RFBR (grant No 19–07–00614).

For citation: M. G. Gonopolskiy and A. B. Glonina, "Research and Development of an Algorithm for the Response Time Estimation in Multiprocessor Systems Under the Interval Uncertainty of the Tasks Execution Times", *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 218-233, 2020.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 27, № 2, 2020
сайт журнала: www.mais-journal.ru

Алгоритм оценки максимального времени отклика задач в многопроцессорных системах с интервальной неопределенностью длительности выполнения работ

М. Г. Гонопольский[1], А. Б. Глонина[1]

DOI: 10.18255/1818-1015-2020-2-218-233

[1]Московский государственный университет им. М. В. Ломоносова, Ленинские горы, д. 1, г. Москва, 119991 Россия.

УДК 519.6
Научная статья
Полный текст на русском языке

Получена 11 февраля 2020 г.
После доработки 9 марта 2020 г.
Принята к публикации 11 марта 2020 г.

В данной статье предложен алгоритм оценки максимального времени отклика задач (Worst Case Response Time — WCRT) в многопроцессорных системах с планировщиками с приоритетом и вытеснением и интервальной неопределенностью длительности выполнения работ. Между задачами имеются зависимости по данным. Для каждой задачи задан приоритет, период и интервал [BCET, WCET], которому принадлежит время выполнения на процессоре работ этой задачи. Если уменьшение времени выполнения задачи A может привести к увеличению времени отклика задачи B, то задача A называется аномальной задачей для задачи B. Согласно выбранному авторами подходу, для получения оценки WCRT некоторой задачи необходимо выполнить два шага. На первом шаге с помощью предложенного в работе алгоритма осуществляется построение множества задач, аномальных для заданной задачи. Приведено доказательство корректности этого алгоритма. На втором шаге с помощью генетического алгоритма выполняется поиск WCRT. Пространство поиска — множество всевозможных наборов длительностей выполнения аномальных задач. Было разработано инструментальное средство на языке Python3, реализующее предложенный подход. Проведено экспериментальное исследование, в ходе которого предложенный метод сравнивался по точности и скорости с двумя известными методами оценки WCRT: методом, не учитывающим интервальную неопределенность, т.е. предполагающим, что время выполнения всех работ равно WCET, а также с переборным методом. Экспериментальное исследование показало, что в отличие от переборного метода, предложенный метод применим для анализа вычислительных систем реальной размерности, а также позволяет достичь большей точности, чем метод, не учитывающий интервальную неопределенность.

Ключевые слова: генетический алгоритм; многопроцессорные системы; наихудшее время отклика; анализ времени отклика; аномалии планирования; планирование вычислений.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Марк Геннадьевич Гонопольский
автор для корреспонденции
orcid.org/0000-0002-0670-8603. E-mail: gmarkmgw@yandex.ru
студент.

Алевтина Борисовна Глонина
автор для корреспонденции
orcid.org/0000-0001-8716-4128. E-mail: alevtina@lvk.cs.msu.su
программист.

Финансирование: Работа выполнена при финансовой поддержке РФФИ (грант № 19–07–00614).

Для цитирования: M. G. Gonopolskiy and A. B. Glonina, “Research and Development of an Algorithm for the Response Time Estimation in Multiprocessor Systems Under the Interval Uncertainty of the Tasks Execution Times”, *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 218-233, 2020.

Введение

В данной статье рассматриваются многопроцессорные системы, представляющие собой набор процессоров, на каждом из которых выполняется набор задач под управлением динамического планировщика со статическими приоритетами и вытеснением. Процессоры соединены некоторой средой передачи данных. Все задачи периодические, в течение каждого периода должен быть выполнен один экземпляр задачи, называемый работой. Каждая задача характеризуется интервалом длительности выполнения работ, приоритетом на сопоставленном ей процессоре и периодом. Между задачами с одинаковым периодом могут существовать зависимости по данным: работа задачи-отправителя посылает сообщения работам задач-получателей и работа задачи-получателя не может начать свое выполнение, пока не получит данные от всех работ задач-отправителей.

Под конфигурацией вычислительной системы понимается совокупность числовых характеристик и взаимосвязей ее компонентов: число процессоров, привязка к ним задач, характеристики задач и зависимости по данным между ними.

Будем называть временем отклика (RT — Response Time) задачи величину, равную максимальному по всем работам этой задачи значению времен их завершения относительно начала периода. Наихудшее время отклика (WCRT — Worst Case Response Time) задачи — максимально возможное время отклика задачи для данной конфигурации системы.

WCRT используется для решения задач, возникающих при проектировании систем жесткого и мягкого реального времени. В частности, значения WCRT необходимы для проверки соблюдения директивных сроков для систем жесткого реального времени [1] и вычисления величины запаздывания для систем мягкого реального времени [2].

Большинство подходов к оценке WCRT основано на предположении, что длительность выполнения каждой работы на процессоре фиксирована и равна максимально возможной [3, 4] (в дальнейшем такой сценарий функционирования систем будем называть *базовым*). Однако этот подход может приводить к заниженным оценкам WCRT задач. Рассмотрим пример, представленный на Рис. 1 и Рис. 2.

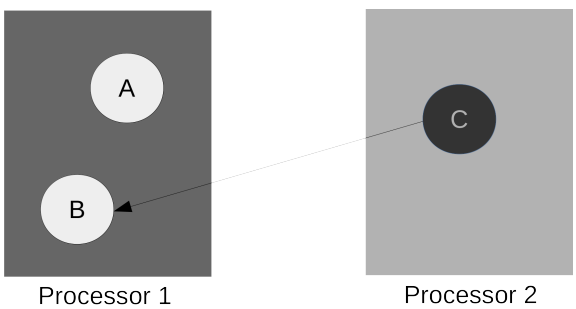


Fig. 1. Scheduling anomaly. Binding tasks to processors and data dependencies

Рис. 1. Пример аномалии планирования. Привязка задач к процессорам и зависимости по данным

Пусть имеется многопроцессорная система, представленная на Рис. 1. Приоритеты задач A и B, привязанных к первому процессору, равны 1 и 2 соответственно. Приоритет задачи C, привязанной ко второму процессору, равен 1. Все задачи имеют одинаковые интервалы длительности выполнения их работ — $[0, 2]$. На левой части Рис. 2 приведен базовый сценарий функционирования системы, то есть сценарий, согласно которому длительность выполнения каждой работы равна максимально возможной длительности (2 единицы времени для задач A, B, C). В этом сценарии работа задачи A успевает завершиться до постановки на выполнение работы задачи B, и WCRT задачи A равно 2.

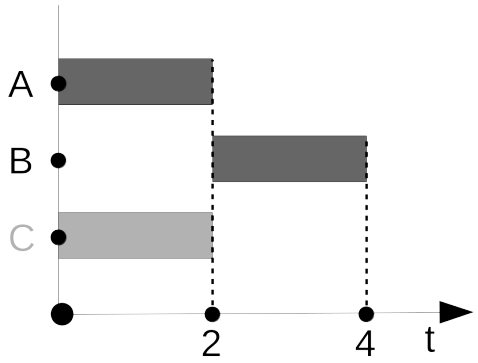


Fig. 2. Scheduling anomaly. Task A completion time increase due to task C execution time reduction

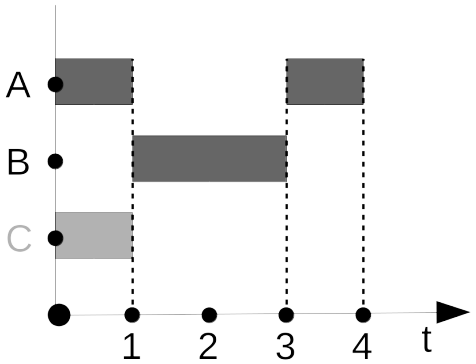


Рис. 2. Пример аномалии планирования. Увеличение времени завершения работы задачи A из-за уменьшения длительности выполнения работы задачи C

На правой части Рис. 2 приведен сценарий функционирования системы, согласно которому длительность выполнения работы задачи C равна 1, а длительности выполнения работ задач B и A по-прежнему равны максимально возможным — 2. В этом сценарии в момент времени 0 начинает выполняться работа задачи A. Работа задачи C завершается в момент времени 1 и отправляет данные работе задачи B, после чего задача B становится готовой к выполнению. Так как задача A имеет приоритет меньший, чем задача B, происходит вытеснение работы задачи A, и на выполнение ставится работа задачи B. Работа задачи B завершает свое выполнение в момент времени, равный 3, после чего происходит возобновление выполнения работы задачи A. Таким образом, работа задачи A завершается в момент времени 4 и WCRT задачи A равно 4.

Приведенный пример демонстрирует, что уменьшение длительности выполнения работы одной задачи может привести к увеличению WCRT другой задачи. И WCRT может достигаться на сценарии функционирования системы, отличном от базового. Поэтому целесообразно применение подходов, учитывающих интервальную неопределенность длительности выполнения работ.

Задачу A будем называть аномальной для задачи B, если уменьшение времени выполнения задачи A может привести к увеличению времени отклика задачи B. Формальное определение аномальной задачи будет введено в разделе 2.

Известен ряд точных методов поиска WCRT, учитывающих интервальную неопределенность длительности выполнения работ. В общем случае сложность всех этих методов экспоненциально зависит от количества задач в системе. Примером таких методов является метод целочисленного линейного программирования (ЦЛП) [5, 6]. Также стоит отметить группу точных методов, основанных на верификации моделей систем, в частности методы, использующие математический аппарат временных автоматов [7, 8]. Еще один точный метод, основанный на полном переборе всех возможных длительностей выполнения задач, приведен в [6]. Из-за экспоненциальной сложности точные методы непригодны для анализа WCRT для систем реальной размерности.

Существуют приближенные методы оценки WCRT [9—13], которые учитывают интервальную неопределенность длительности выполнения работ, но дают завышенную оценку, недостижимую на практике.

Еще один метод оценки WCRT был предложен в работе [14]. В этой работе авторы решают задачу оценки WCRT в три этапа:

1. Нахождение подмножества аномальных задач, для этого используется метод Раку и Эрнста [15].
2. Редуцирование интервалов длительностей выполнения найденных задач из пункта 1.
3. Поиск оценки времени отклика задач с использованием генетического алгоритма.

Однако метод Раку и Эрнста не находит все возможные аномальные задачи. В частности при использовании данного метода для примера, представленного на Рис. 1 и Рис. 2, задача C не будет отмечена как аномальная для задачи A.

Таким образом, все известные методы обладают теми или иными недостатками, поэтому авторами было принято решение разработать новый метод оценки WCRT, основанный на [14]. Предлагаемый метод оценки WCRT для заданной задачи состоит в последовательном выполнении двух шагов:

1. Построение множества задач, аномальных для данной задачи; для этого был разработан новый алгоритм поиска аномальных задач, корректность которого была строго доказана.
2. Поиск WCRT с помощью генетического алгоритма.

Стоит отметить, что получаемая оценка является нижней, однако она имеет большую точность, чем оценки, получаемые с помощью методов, основанных на анализе базового сценария функционирования системы. Поэтому использование предложенного метода для решения задач, возникающих при проектировании систем реального времени, вместо методов, основанных на анализе базового сценария, позволяет сократить количество ошибок при проектировании.

В отличие от методов, имеющих экспоненциальную сложность, предложенный метод применим для анализа систем реальной размерности.

В отличие от приближенных методов, дающих оценки сверху, предложенный метод позволяет найти гарантированно достижимую оценку WCRT, а также сценарий функционирования системы, на котором эта оценка достигается. Поэтому предложенный метод может использоваться для анализа пессимистичности приближенных оценок сверху: получив достижимую нижнюю оценку и убедившись в том, что она слабо отличается от верхней оценки, можно пользоваться последней, не опасаясь, что оценка сильно завышена.

1. Формальная постановка задачи

Пусть дана многопроцессорная система, содержащая N одинаковых процессоров.

Рабочая нагрузка представляется в виде кортежа $WL = (T, M)$, где $T = \{t_i\}$ — набор задач, $M = \{M_i\}$ — набор сообщений, передаваемых между задачами.

Каждая задача представляется в виде кортежа $t_i = ([bcet_i, wcet_i], H_i, P_i) \in T$, где:

- $[bcet_i, wcet_i]$ $(bcet_i, wcet_i \in \mathbb{N})$ — интервал длительности выполнения работ задачи;
- $H_i \in \mathbb{N}$ — период;
- $P_i \in \mathbb{N}$ — уникальный в рамках процессора приоритет.

Каждое сообщение представляется в виде кортежа $M_i = (S_i, R_i, dn_i)$, где:

- $S_i \in T$ — задача-отправитель;
- $R_i \in T$ — задача-получатель;
- $dn_i \in \mathbb{N}$ — длительность передачи сообщения.

Рабочей нагрузке соответствует ациклический направленный граф, далее обозначаемый как G; вершины в нем соответствует задачам, а дуги — сообщениям. Для любой пары вершин в графе G верно, что эти вершины могут быть связаны только одной дугой.

Отображение $Bind : T \longrightarrow \{1, ..., N\}$ задает привязку задач к процессорам.

Конфигурация вычислительной системы — это двойка $(WL, Bind)$, где WL — описание рабочей нагрузки, $Bind$ — привязка задач к процессорам.

Пусть $CONF$ — множество всевозможных конфигураций многопроцессорной системы. При анализе некоторой конфигурации $conf \in CONF$ многопроцессорной системы рассматривается функционирование этой системы в течение интервала планирования длительности L. Как правило, L равно наименьшему общему кратному периодов задач из T.

Для каждой задачи $t_i \in T$ на L определен набор работ $W_i = \{w_{ij}\}$, где $j \in \overline{1, [\frac{L}{P_i}]}$. Пусть $t_i \in T$ — исследуемая задача, Out_{ij} — время завершения ее j-й работы ($j/ge1$). Тогда время отклика R_{ij} этой работы определяется следующим образом:

$$R_{ij} = (Out_{ij} - (j-1) * H_i). \tag{1}$$

Время отклика R_i задачи t_i определяется следующим образом:

$$R_i = \max_j R_{ij}. \tag{2}$$

Пусть $\tau = (\tau_1, ..., \tau_{|T|})$ — набор длительностей выполнения задач для заданной конфигурации, определяющий конкретный сценарий функционирования системы; $\tau_i \in (bcet_i, wcet_i)$.

Пусть $C(conf)$ — множество всевозможных наборов τ для конфигурации $conf$. Тогда задача поиска WCRT для заданной задачи этой конфигурации имеет следующую формальную постановку.

Дано: конфигурация многопроцессорной системы $conf \in CONF$ и задача $t_i \in T$.

Найти:

$$\max_{C(conf)} R_i. \tag{3}$$

2. Алгоритм поиска аномальных задач

Согласно предложенному подходу для получения оценки WCRT заданной задачи необходимо сначала найти множество задач, аномальных для этой задачи. В данном разделе приведено описание разработанного авторами алгоритма поиска аномальных задач, а также обоснование его корректности.

2.1. Теоретическое обоснование алгоритма

Введем ряд определений, необходимых для описания алгоритма поиска аномальных задач.

Согласно разделу 1, рабочей нагрузке соответствует ациклический направленный граф G, вершины в котором взаимно однозначно соответствуют задачам, а дуги — сообщениям. Поскольку для любой пары вершин в графе G верно, что эти вершины могут быть связаны только одной дугой, то в качестве *пути* в графе G можно рассматривать последовательность вершин, в которой каждая вершина соединена дугой со следующей.

В данном разделе для краткости под задачей, находящейся в некотором пути, будем понимать вершину, соответствующую этой задаче в графе G и находящуюся в этом пути.

Нижним путем задачи A назовем путь в графе G, начинающийся с задачи T, которая непосредственно зависит по данным от A, и заканчивающийся задачей P, от которой не зависит по данным ни одна задача.

Верхним путем задачи A назовем путь в графе G, заканчивающийся такой задачей T, что A непосредственно зависит по данным от T, и начинающийся с задачи P, которая не зависит по данным ни от каких задач.

Если задача A и задача B находятся на одном процессоре и не имеют зависимостей друг от друга, то будем говорить, что между задачей B и задачей A имеется *фиктивная* связь.

Два пути будем называть связанными фиктивной связью, если между задачей, которой оканчивается один из этих путей, и задачей, с которой начинается второй путь, имеется фиктивная связь.

Задача B *аномальна* для задачи A, если существуют два сценария функционирования системы, отличающиеся друг от друга лишь тем, что в первом сценарии время выполнения работ задачи B больше, чем во втором сценарии, вследствие чего $WCRT(A)$ в первом сценарии имеет значение меньшее, чем во втором.

Задачу B назовем *аномальной задачей первого типа* для выбранной задачи A, если существуют два сценария функционирования системы, отличающиеся друг от друга лишь тем, что в первом сценарии время выполнения работ задачи B больше, чем во втором сценарии, и существует такая задача P из верхнего пути задачи A, некоторая работа которой во втором сценарии завершается позже, чем в первом сценарии, вследствие чего $WCRT(A)$ во втором сценарии больше, чем в первом сценарии. Обозначим множество аномальных задач первого типа задачи A как $IT(A)$ (Inheritable Tasks).

Задачу B назовем *аномальной задачей второго типа* для выбранной задачи A, если существуют два сценария функционирования системы, отличающиеся друг от друга лишь тем, что в первом сценарии время выполнения работ задачи B больше, чем во втором сценарии, и существует такая задача P, имеющая больший приоритет, чем задача A,находящаяся с ней на одном процессоре, и некоторая работа которой во втором сценарии начинает выполняться раньше, чем в первом сценарии, вследствие чего $WCRT(A)$ во втором сценарии больше, чем в первом сценарии. Обозначим множество аномальных задач второго типа задачи A как $HT(A)$ (Hidden Tasks).

Задачу B назовем *аномальной задачей третьего типа* для выбранной задачи A, если существуют два сценария функционирования системы, отличающиеся друг от друга лишь тем, что в первом сценарии время выполнения работ задачи B больше, чем во втором сценарии, и существует такая задача P, имеющая больший приоритет, чем задача A, находящаяся с ней на одном процессоре, и некоторая работа которой во втором сценарии начинает выполняться позже, чем в первом сценарии, вследствие чего $WCRT(A)$ во втором сценарии больше, чем в первом сценарии. Обозначим множество аномальных задач третьего типа задачи A как $HT(A)$ (Separate Tasks).

Лемма 1. *Задача B является аномальной задачей для задачи A тогда и только тогда, когда она принадлежит $IT(A) \cup HT(A) \cup ST(A)$.*

Доказательство. Докажем необходимость и достаточность.

1. Необходимость. Пусть задача B является аномальной для задачи A, то есть существуют два таких сценария функционирования системы, отличающиеся друг от друга лишь тем, что в первом сценарии время выполнения работ задачи B больше, чем во втором сценарии, и $WCRT(A)$ в первом сценарии имеет значение меньшее, чем во втором. Следовательно, имеет место один из четырех случаев:
   (a) Во втором сценарии из-за более раннего, по сравнению с первым сценарием, окончания работы некоторой задачи B закончится позже работа задачи, от которой зависит задача A. В этом случае задача $B \in IT(A)$, исходя из определения множества аномальных задач первого типа.
   (b) Во втором сценарии работу задачи A вытеснит работа другой, более приоритетной задачи P, из-за более раннего, по сравнению с первым сценарием, окончания работы некоторой задачи B. В этом случае работа задачи P либо закончится позже, по сравнению с первым сценарием, из чего следует, что $B \in ST(A)$ по определению множества аномальных задач третьего типа, либо работа задачи P начнется раньше, чем в первом сценарии, из чего следует, что задача $B \in HT(A)$, по определению множества аномальных задач второго типа.
   (c) Во втором сценарии работа задачи A будет вынуждена ждать окончания работы более приоритетной задачи P, которая начнет выполняться раньше из-за более раннего,

по сравнению с первым сценарием, окончания работы некоторой задачи B. То есть в первом сценарии задача A успеет завершиться до начала задачи P, а в втором — нет. В этом случае $B \in HT(A)$.

(d) Во втором сценарии работа задачи A будет вынуждена ждать окончания работы более приоритетной задачи P, которая начнет выполняться позже, чем в первом сценарии, из-за более раннего окончания, по сравнению с первым сценарием, работы некоторой задачи B. То есть в обоих сценариях задача A становится готовой после начала выполнения задачи P и чем позднее начнет выполняться P, тем позднее начнет выполняться A. В этом случае $B \in ST(A)$.

2. Достаточность. Любая задача, содержащаяся во множестве $IT(A) \cup HT(A) \cup ST(A)$ является аномальной, исходя из определения этих множеств.

□

Пусть $Dep(A)$ — множество всех задач, от которых задача A имеет непосредственную зависимость по данным.

Лемма 2. *Множество аномальных задач первого типа для выбранной задачи A исчерпывается объединением множеств аномальных задач для задач, от которых задача A имеет непосредственную зависимость по данным.*

Доказательство. По определению множество аномальных задач первого типа для задачи A — это все такие задачи, для каждой из которых существуют два сценария функционирования системы, отличающиеся друг от друга лишь тем, что в первом сценарии время выполнения работ соответствующей задачи больше, чем во втором сценарии, и существует такая задача P из верхнего пути задачи A, работы которой во втором сценарии завершаются позже, чем в первом сценарии. Иными словами, это аномальные задачи для задач из верхних путей задачи A. Однако в выбранном пути каждая последующая задача имеет среди своих аномальных задач первого типа аномальные задачи задач, предшествующих ей в пути. Следовательно, чтобы рассмотреть объединение множеств аномальных задач для задач из верхних путей задачи A, достаточно рассмотреть лишь объединение множеств аномальных задач для задач из $Dep(A)$. □

Пусть $I_1(A)$ — все более приоритетные задачи, чем A и находящиеся с ней на одном процессоре. $S(A) \equiv I_1(A) \cup Dep(A)$,

Пусть $M = \{z_i\}_{i=1}^{m}$ — некоторое множество задач. Обозначим как $U(M)$ следующее множество: $U(M) \equiv S(z_1) \cup ... \cup S(z_m)$. Обозначим как $D(M)$ следующее множество: $D(M) \equiv Dep(z_1) \cup ... \cup Dep(z_m)$. Положим $E_0(M) \equiv U(M) \cup M$.

Определим $E_{i+1}(M)$ как $E_{i+1}(M) \equiv U(E_i(M)) \cup E_i(M)$. Тогда, так как множество задач в системе конечно, $\exists n > 0 \in \mathbb{N}$: $\forall i \geq n\ E_i(M) = E_n(M)$. $E_n(M)$ будем обозначать как $E_{comp}(M)$

Лемма 3. *Множество $ST(A) \cup HT(A)$ для задачи A содержится во множестве $E_{comp}(D(I_1(A)))$.*

Доказательство. Пусть A — рассматриваемая задача. Предположим, что задача $B \notin E_{comp}(D(I_1(A)))$ является аномальной задачей второго или третьего типа для задачи A, то есть $B \in ST(A) \cup HT(A)$. Тогда из определения аномальной задачи второго типа и определения аномальной задачи третьего типа следует, что в результате раннего завершения работы задачи B происходит раннее или позднее завершение работы некоторой задачи $P \in I_1(A)$. Поэтому задача B принадлежит конкатенации путей, связанных фиктивными связями и заканчивающейся задачей P. Тогда задача $B \in E_{comp}(D(I_1(A)))$, поскольку будет найдено такое $k > 0 \in \mathbb{N}$: $B \in E_k(I_1(A))$, что противоречит введенному предположению. Следовательно, предположение неверно, и лемма доказана.

□

Множество всех задач из верхних путей задачи A будем обозначать как $Over(A)$.

Для краткости под глубиной задачи в графе G будем понимать максимальную длину пути до вершины в графе G, соответствующей данной задаче.

Лемма 4. *$IT(A)$ для задачи A содержится в $\cup_{P \in Over(A)} (E_{comp}(D(I_1(P))))$.*

Доказательство. Воспользуемся методом математической индукции. Пусть задана некоторая задача A графа G, с глубиной $n \in \mathbb{N}$.

1. Пусть $n = 1$, тогда задача A не имеет аномальных задач первого типа ($IT(A) = \varnothing$), поскольку не имеет зависимостей по данным. Следовательно, утверждение верно.
2. Пусть для $n = k$ утверждение верно. Докажем утверждение для $n = k + 1$. Согласно лемме 1 и лемме 2, $IT(A_{k+1}) = \cup_{P \in Dep(A_{k+1})}(IT(P) \cup HT(P) \cup ST(P))$.
   Исходя из предположения индукции, для каждой такой задачи $P \in Dep(A_{k+1})$: $IT(P)$ содержится в $\cup_{J \in Over(P)} (E_{comp}(D(I_1(J))))$. Далее, $HT(P) \cup ST(P)$ содержится в $E_{comp}(D(I_1(P)))$, согласно лемме 3. Следовательно, утверждение леммы 4 верно.

□

Теорема 1. *Множество аномальных задач для задачи A содержится в множестве $E_{comp}(D(I_1(A))) \cup (\cup_{P \in Over(A)} (E_{comp}(D(I_1(P)))))$.*

Доказательство. Утверждение теоремы напрямую следует из леммы 1, леммы 3 и леммы 4. □

Таким образом, для получения множества задач, аномальных для заданной задачи A, необходимо выполнить следующие действия:

1. Найти множество $\cup_{P \in Over(A)} (E_{comp}(D(I_1(P))))$, далее обозначаемое как $CIT(A)$, и множество $E_{comp}(D(I_1(A)))$.
2. Убрать из $E_{comp}(D(I_1(A))) \cup CIT(A)$ все задачи, которые не являются аномальными для рассматриваемой задачи.

На данный момент авторам не известен общий метод определения, является ли некоторая задача из множества $E_{comp}(D(I_1(A))) \cup CIT(A)$ аномальной для задачи A. Однако авторами был разработан алгоритм (алгоритм перекрестия), позволяющий исключить из множества $E_{comp}(D(I_1(A))) \cup CIT(A)$ ряд задач, гарантированно не являющихся аномальными.

Для задач A и B задано *перекрестие*, если существует такой нижний путь задачи A, в который входит задача B. В таком случае работа задачи B не может начать свое выполнение, пока не закончит свое выполнение работа задачи A. Следовательно, уменьшение времени выполнения работ задачи B не может повлиять на время отклика задачи A, и задача B не является аномальной для задачи A.

Исключив с помощью алгоритма перекрестия из множества $E_{comp}(D(I_1(A))) \cup CIT(A)$ ряд неаномальных задач, получим множество, полностью содержащее все аномальные задачи для рассматриваемой задачи, однако, возможно, содержащее некоторые лишние задачи. Это не приведет к получению некорректных результатов, но может увеличить длительность работы генетического алгоритма, используемого для поиска WCRT.

Для удобства вместо $E_{comp}(D(I_1(A)))$ будем использовать множества:

- $I_2(A)$ — множество всех задач, входящих в верхние пути задач из $I_1(A)$;
- $I_3(A)$ — множество $\cup_{P \in I_2(A)} (E_{comp}(I_1(P))$.

Лемма 5. *Множество $I_2(A) \cup I_3(A)$ для задачи A совпадает с $E_{comp}(D(I_1(A)))$.*

Доказательство. Пусть A — некоторая задача.

1. Докажем, что для любой задачи $B \in I_2(A)$ верно, что $B \in E_{comp}(D(I_1(A)))$.
   M — некоторое множество задач. Положим $Dep_0(M) \equiv D(M)$.
   Определим $D_{i+1}(M)$ как $D_{i+1}(M) \equiv Dep(D_i(M)) \cup D_i(M)$.
   Тогда, так как множество задач в системе конечно, $\exists\ n > 0 \in \mathbb{N}$: $\forall i \geq n\ D_i(M) = D_n(M)$. $D_n(M)$ будем обозначать как $D_{comp}(M)$. Возьмем в качестве M множество $D(I_1(A))$ и получим множество $D_{comp}(D(I_1(A))) = I_2(A)$. По построению $E_{comp}(M) = E_n(M)$, такое что $E_i(M) = E_n(M)$, для любого $i > n$, где $E_{i+1}(M) = U(E_i(M)) \cup E_i(M)$, $E_0(M) = U(M) \cup M$. По определению $U(M) = (I_1(z_1) \cup Dep(z_1)) \cup ... \cup (I_1(z_m) \cup Dep(z_m))$, $D(M) = Dep(z_1) \cup ... \cup Dep(z_m)$, $z_i \in M$, $|M| = m$. Следовательно, $D(M) \in U(M)$ и $D_i(M) \in E_i(M)$. Поэтому $D_{comp}(D(I_1(A))) \in E_{comp}(D(I_1(A)))$.
2. Докажем, что для любой задачи $B \in I_3(A)$ верно, что $B \in E_{comp}(D(I_1(A)))$.
   Пусть $P \in I_2(A)$, $B \in E_{comp}(I_1(P))$. Тогда существует такая, имеющая фиктивную связь с задачей P, задача C (может быть, совпадающая с задачей B), на которую оканчивается конкатенация соединенных фиктивными связями путей, содержащая задачу B. Из рассуждений предыдущего пункта следует, что найдется такое k, что $P \in D_k(D(I_1(A)))$. Следовательно, по построению множества $E_{comp}(D(I_1(A)))$, множество $E_{comp}(S(P))$ полностью содержится во множестве $E_{comp}(D(I_1(A)))$. Так как множество $S(P) = I_1(P) \cup Dep(P)$, тогда множество $E_{comp}(I_1(P))$ содержится во множестве $E_{comp}(S(P))$. Следовательно, $E_{comp}(I_1(P))$ содержится в $E_{comp}(D(I_1(A)))$. Следовательно, $B \in E_{comp}(D(I_1(A)))$.
3. Докажем, что для любой задачи $T \in E_{comp}(D(I_1(A)))$ верно, что $T \in I_3(A) \cup I_2(A)$.
   Напомним, что $I_3(A) = \cup_{P \in I_2(A)} (E_{comp}(I_1(P))$. Проведем доказательство от противного. Предположим, что существует некоторая задача $T \in E_{comp}(D(I_1(A)))$ такая, что $T \notin \cup_{P \in I_2(A)} (E_{comp}(I_1(P)) \cup I_2(A)$. Тогда, по построению множества $E_{comp}(D(I_1(A)))$, возможны два случая:
   (a) Задача T совпадает с некоторой задачей N из $D(I_1(A))$, тогда $T \in I_2(A)$, что противоречит введенному предположению.
   (b) Задача T принадлежит конкатенации путей, связанных фиктивными связями и заканчивающейся задачей N. Тогда существует процессор, на котором наблюдается фиктивная связь между некоторой задачей C_1 из верхнего пути задачи N (или совпадающей с ней) и некоторой отличной от C_1 задачей C_2, которой оканчивается конкатенация путей, связанных фиктивными связями, содержащая задачу T. Из этого следует, что задача $T \in E_{comp}(I_1(C_1))$, так как найдется такое k, что $T \in E_k(I_1(C_1))$. Следовательно, $T \in \cup_{P \in I_2(A)} (E_{comp}(I_1(P))$, что противоречит введенному предположению.
   Таким образом, $T \in I_3(A) \cup I_2(A)$, что и требовалось доказать.

□

Использование такого разбиения упрощает реализацию алгоритма, описанного далее. Удобно найти множество $I_2(A)$ в виде верхних путей задач из $I_1(A)$ и на основании этого множества найти $I_3(A)$.

Итого, для получения множества, содержащего все аномальные задачи для заданной задачи A необходимо найти $I_1(A) \cup I_2(A) \cup I_3(A)$, то есть $(\cup_{P \in Over(A)} (E_{comp}(D(I_1(P))))) \cup E_{comp}(D(I_1(A)))$, а затем исключить из этого множества задачи, заведомо не являющиеся аномальными, согласно алгоритму перекрестия. Корректность предложенного алгоритма поиска аномальных задач (а именно тот факт, что полученное множество будет содержать все задачи, аномальные для заданной задачи A) следует из теоремы 1 и леммы 5.

2.2. Описание алгоритма в виде псевдокода

Уровнями графа G назовем множества задач, каждое из которых содержит лишь задачи, имеющие одинаковую глубину в графе G. Например, *уровень ноль* содержит лишь задачи, не имеющие

зависимостей; *уровень один* — задачи, имеющие зависимости только от задач уровня ноль и т.д. Разбиение множества задач на уровни выполняется при поиске аномальных задач первого типа.

Введем обозначения для вспомогательных функций:

- UnderLine(A) — функция, возвращающая множество всех нижних путей задачи A.
- OverLine(A) — функция, возвращающая множество всех верхних путей задачи A.
- Getanomaltasks(A) — функция, возвращающая множество всех известных на данный момент аномальных задач для задачи A.
- Dep(A) — функция, возвращающая множество $Dep(A)$, то есть множество всех задач, от которых задача A непосредственно имеет зависимость по данным.
- CoreNum($task$) — функция, возвращающая номер процессора, на котором выполняется задача $task$.
- TasksFromCore(HW) — функция, возвращающая множество задач, работы которых выполняются на процессоре с номером HW.
- Sop(A) — функция, возвращающая множество $S(A)$, то есть множество всех задач, с которыми задача A имеет фиктивную связь, и множество всех задач, от которых задача A непосредственно имеет зависимость по данным.
- GetLayers(G) — функция, возвращающая упорядоченное по возрастанию номеров множества уровней.
- I1(A) — функция, возвращающая $I_1(A)$ для задачи A, то есть множество задач, выполняющихся на одном процессоре с задачей A и имеющих приоритет, больший, чем у задачи A.
- Depend(M) — функция, возвращающая $D(M)$ для множества M.
- Priority(M) — функция, возвращающая приоритет задачи A на соответствующем ей процессоре.

Приведем описания на псевдокоде функций, входящих в состав алгоритма поиска аномальных задач.

Алгоритм нахождения перекрестия между задачами T и S реализует функция isCross(T,S). Возвращает значение логического типа данных: если между выбранными задачами есть перекрестие, тогда возвращается *Истина*, иначе — *Ложь*. Функция имеет следующий вид:

```
|bool IsCross(T,S):
|   F := false // флаг наличия перекрестия
|   if T ≠ S:
|      for Path from UnderLine(T)
|         for task from Path:
|            if task = S:
|               F := true // задача не нуждается в рассмотрении
|   else:
|      F := true // задача не нуждается в рассмотрении
|   return F
```

Алгоритм поиска аномальных задач первого типа реализует функция CIT(A). Согласно лемме 1, для того, чтобы найти аномальные задачи первого типа для данной задачи A достаточно выполнить проход по всем задачам, от которых задача A имеет непосредственную зависимость по данным, и объединить все аномальные задачи этих задач в одно множество. В процессе применения алгоритма осуществляется обход графа по уровням — от нулевого до последнего, что позволяет заменить полное вычисление множества $(\cup_{P \in Over(A)} (E_{comp}(D(I_1(P)))))$ обходом задач из множества $Dep(A)$. Функция имеет следующий вид:

```
|set CIT(A):
|   Tasks := ∅
|   for T from Dep(A):
|       Tasks := Tasks ∪ Getanomaltasks(T)
|   return Tasks
```

Алгоритм поиска $E_{comp}(M)$ для множества $\notin \equiv D(I_1(A))$, для некоторой задачи A, реализует функция Ecomp(M). Как упоминалось ранее, множество $E_{comp}(M)$ для некоторого множества M строится путем объединения множеств $I_2(A)$ и $I_3(A)$. Таким образом, для каждой задачи $B \in I_1(A)$ требуется найти множество верхних путей, затем для каждой задачи C из найденных верхних путей требуется провести операцию поиска соответствующей части множества $I_3(A)$, представляющую из себя $E_{comp}(S(I_1(C))) \cup I_1(C)$. Функция имеет следующий вид:

```
|set Ecomp(M):
|   AnomalSet := ∅
|   for T from M:
|       for OverT from OverLine(T):
|           TmpSet := ∅
|           for Task from OverT:
|               AnomalSet.add(Task) // поиск I2
|               HW := CoreNum(Task)
|               CoreTasks := TasksFromCore(HW)
|               for Hidd from CoreTasks:
|                   if Priority(Hidd) ≤ Priority(Task):
|                       CoreTasks := CoreTasks\Hidd // удалить задачу из множества
|               for Hidd from CoreTasks:
|                   TmpSet := TmpSet ∪ Hidd
|                   TmpSet := Ecomp(Sop(Hidd)) ∪ DummySet // поиск I3
|           AnomalSet := TmpSet ∪ AnomalSet
|   return AnomalSet
```

Алгоритм поиска аномальных задач реализует функция FindAnomalTasks(G). Эта функция возвращает массив множеств $E_{comp}(D(I_1(A))) \cup CIT(A)$ для каждой задачи A в системе, описываемой графом G. Функция имеет следующий вид:

```
|set array FindAnomalTasks(G):
|   Layers := GetLayers(G) // разбить граф на уровни
|   for Layer from Layers:
|       for T from Layer:
|           MainSets[T] := CIT(T)
|           Positive := ∅
|           for Susp from I1(T):
|               if IsCross(T,Susp): // перекрестие ⟶ следующая задача
|                   continue
|               else Positive = Positive ∪ Susp
|           MainSets[T] := MainSet[T] ∪ Ecomp(Depend(Positive))
|   return MainSets
```

3. Генетический алгоритм оценки WCRT

Авторами был разработан алгоритм, позволяющий получить оценку WCRT для заданной задачи и заданной конфигурации системы, учитывающий влияние аномальных задач. За основу была взята идея из [14]: оценка максимального времени отклика задач проводится с помощью генетического алгоритма.

За основу взят классический генетический алгоритм [16].

Пусть задана некоторая конфигурация системы и задача T, для которой требуется найти оценку WCRT. *Особь* представляет собой последовательность генов. Количество генов равно количеству задач, аномальных для задачи T. Каждый ген соответствует некоторой задаче A, аномальной для задачи T, и имеет целочисленное значение из интервала длительности выполнения задачи A. В том числе значение гена может быть равно одной из границ этого интервала.

Функция приспособленности особи — время отклика задачи T, вычисленное для случая, когда длительность выполнения каждой аномальной задачи равна значению соответствующего гена, а длительности выполнения остальных задач равны правым границам соответствующих интервалов. Для вычисления функции приспособленности используется средство, описанное в работе [17]. Авторами этой работы предложен метод получения временной диаграммы функционирования вычислительной системы с заданной конфигурацией при фиксированных длительностях выполнения задач. Временная диаграмма содержит события постановки работ на выполнение, события вытеснения и завершения работ. Значение времени отклика задачи получается посредством анализа временной диаграммы согласно формулам, приведенным в разделе 1.

В качестве оператора *скрещивания* используется классическое одноточечное скрещивание. *Мутация* представляет собой изменение некоторого количества генов в особи на случайные значения из соответствующих интервалов длительности выполнения задач. В результате выполнения операторов скрещивания и мутации в популяцию попадают новые особи и ее размер возрастает. В процессе *отбора* формируется новая популяция таким образом, что в нее попадает некоторое количество лучших особей текущей популяции, а также для обеспечения разнообразия, некоторое количество худших особей. При этом итоговый размер новой популяции равен размеру предыдущей популяции до выполнения скрещивания и мутации.

Критерием останова является отсутствие изменения функции приспособленности лучшей особи в течение определенного числа итераций. При возникновении ситуации, в которой итоговый результат работы алгоритма меньше, чем WCRT в базовом сценарии функционирования системы, в качестве результата берется WCRT для базового сценария.

4. Экспериментальное исследование

Предложенный авторами статьи метод сравнивался с методом, основанном на анализе базового сценария функционирования системы, и с методом, основанном на полном переборе. Критериями сравнения выступали время и точность работы алгоритма.

Для проведения экспериментов было разработано инструментальное средство на языке Python3, реализующее предложенный в разделах 2 и 3 подход. Также реализован метод полного перебора и метод, основанный на анализе базового сценария функционирования системы. Кроме того разработано средство генерации конфигураций систем различной размерности.

Эксперименты проводились на данных, соответствующих реальной системе, и на искусственно сгенерированных данных.

Поскольку истинное значение WCRT можно определить за приемлемое время лишь для конфигураций небольшой размерности, сравнение точности генетического алгоритма с точностью метода, основанного на базовым сценарии, было выполнено лишь для таких конфигураций.

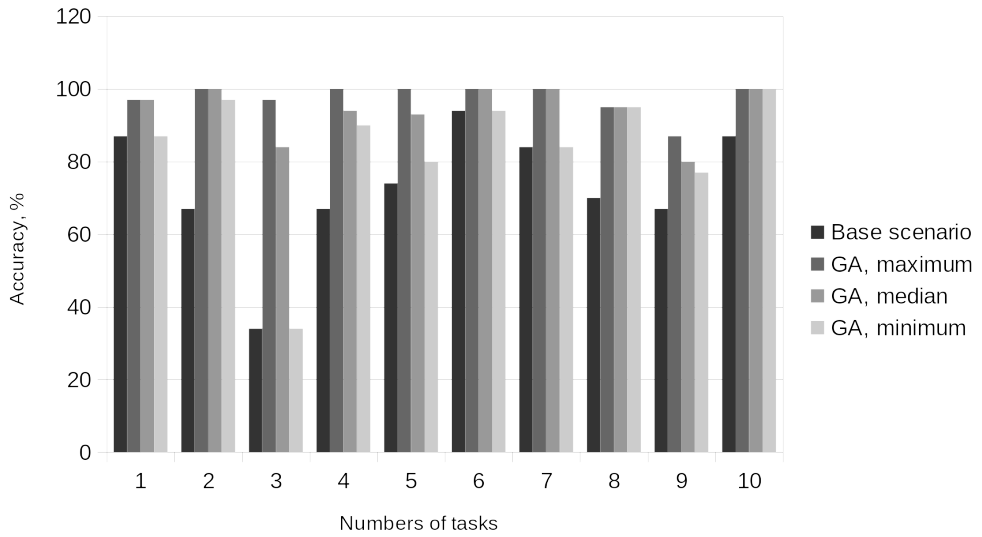


Fig. 3. Comparison of the minimum, median and maximum accuracy of the genetic algorithm with the method based on the base scenario analysis. Artificially generated data

Рис. 3. Сравнение минимальной, медианной и максимальной точности генетического алгоритма (GA) и метода, основанного на анализе базового сценария (Base scenario), на искусственно сгенерированных данных

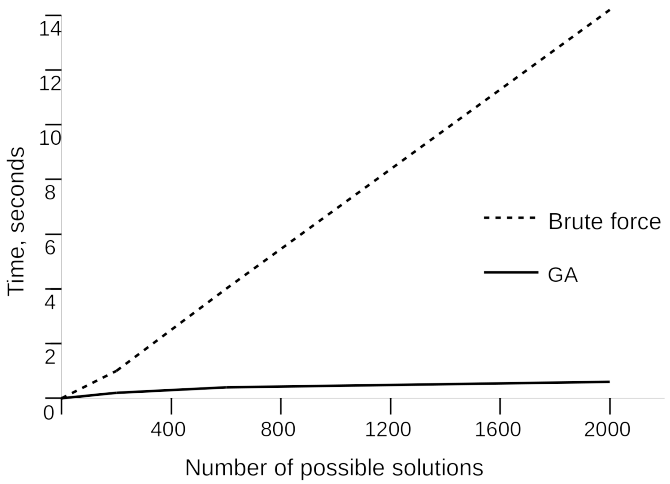


Fig. 4. Comparison of genetic algorithm speed to speed of the brute force depending on the number of possible solutions. Artificially generated data

Рис. 4. Сравнение скорости работы генетического алгоритма (GA) со скоростью работы полного перебора (Brute force) в зависимости от количества возможных решений, на искусственно сгенерированных данных

На Рис. 3 представлены результаты сравнения точности алгоритмов. Во всех экспериментах медианное значение точности генетического алгоритма было не меньше 80%. В 70% случаев алгоритму удалось достигнуть точного значения.

На Рис. 4 представлены результаты сравнения времени работы предложенного генетического алгоритма со временем работы полного перебора на искусственно сгенерированных данных.

Также была проведена серия экспериментов на данных, приближенных к реальным. Конфигурация вычислительной системы, для задач которой производилась оценка WCRT, содержит 164 задачи и 100 сообщений. Результаты этих экспериментов представлены на Рис. 5

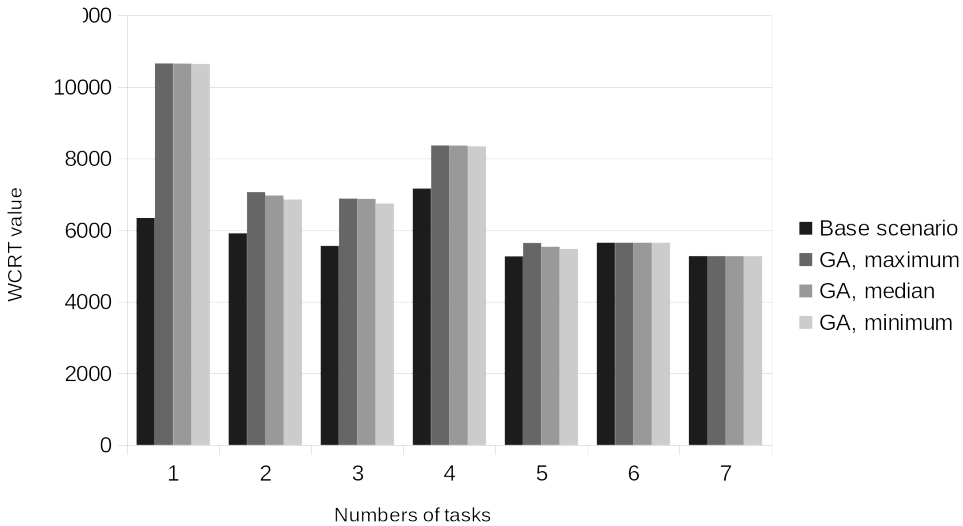


Fig. 5. Comparison of the minimum, median and maximum results of the genetic algorithm with the results of a method based on the base scenario analysis. Realistic data

Рис. 5. Сравнение минимальных, медианных и максимальных результатов генетического алгоритма (GA) и результатов метода, основанного на анализе базового сценария (Base scenario), на данных, приближенных к реальным

Были получены следующие результаты: в зависимости от входных данных точность разработанного алгоритма превышает точность метода, основанного на базовом сценарии в 1.0 — 1.68 раз; при количестве возможных решений, равном 10^{36}, разработанный алгоритм работает в среднем около 12 минут.

5. Заключение

Авторами разработан метод оценки WCRT, заключающийся в последовательном применении двух предложенных ими алгоритмов: алгоритма поиска аномальных задач, позволяющего найти множество задач, включающее в себя все аномальные задачи для рассматриваемой задачи, и алгоритма оценки времени отклика задач на основе множества аномальных задач.

Было выполнено экспериментальное исследование метода на искусственно сгенерированных данных и данных, приближенных к реальным, показавшее, что предложенный метод позволяет повысить точность оценки WCRT по сравнению с методом, не учитывающим интервальную неопределенность (до 1.68 раз на реальных данных).

В качестве направлений дальнейших исследований можно выделить следующие задачи: разработка, исследование и сравнение с ГА других поисковых алгоритмов оценки WCRT, в том числе точных (например, метод ветвей и границ); исследование интервалов длительности выполнения работ для редукции пространства возможных решений; доработка предложенного подхода с целью получения оценок WCRT одновременно для групп задач.

References

[1] N. Holsti, T. Langbacka, and S. Saarinen, "Using a Worst-Case Execution Time Tool for Real-Time Verification of the DEBIE Software", *EUROPEAN SPACE AGENCY-PUBLICATIONS-ESA SP*, vol. 457, pp. 307–312, 2000.

[2] C. Liu and J. H. Anderson, "Task Scheduling with Self-Suspensions in Soft Real-Time Multiprocessor Systems", in *2009 30th IEEE Real-Time Systems Symposium*, 2009, pp. 425–436.

[3] J. C. P. Gutierrez, J. J. G. Garcia, and M. G. Harbour, "Best-Case Analysis for Improving the Worst-Case Schedulability Test for Distributed Hard Real-Time Systems", in *Proceedings of 10th EUROMICRO Workshop on Real-Time Systems*, 1998, pp. 35–44.

[4] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[5] J. Kim and et al., "An ILP-Based Worst-Case Performance Analysis Technique for Distributed Real-Time Embedded Systems", in *Proceedings of IEEE 33rd Real-Time Systems Symposium*, 2012, pp. 363–372.

[6] R. Wilhelm, "Why AI+ ILP is Good for WCET, but MC is not, nor ILP Alone", in *International Workshop on Verification, Model Checking, and Abstract Interpretation*, 2004, pp. 309–322.

[7] A. Brekling, M. R. Hansen, and J. Madsen, "Models and Formal Verification of Multiprocessor System-on-Chips", *The Journal of Logic and Algebraic Programming*, vol. 77, no. 1-2, pp. 1–19, 2008.

[8] J. Kany and S. Madsen, "Design Optimisation of Fault-Tolerant Event-Triggered Embedded Systems", PhD thesis, Doctoral dissertation, Master's thesis, Tech. Univ. of Denmark, Lyngby, Denmark, 2007, 176 pp.

[9] J. Kim and et al., "A Novel Analytical Method for Worst Case Response Time Estimation of Distributed Embedded Systems", in *Proceedings of 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–10.

[10] K. Tindell and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", *Microprocessing and microprogramming*, vol. 40, no. 2-3, pp. 117–134, 1994.

[11] R. I. Davis and et al., "Controller Area Network (CAN) Schedulability Analysis: Refuted, Revisited and Revised", *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.

[12] J. C. Palencia and M. G. Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets", in *Proceedings 19th IEEE Real-Time Systems Symposium*, 1998, pp. 26–37.

[13] O. Redell, "Analysis of Tree-Shaped Transactions in Distributed Real-Time Systems", in *Proceedings of 16th Euromicro Conference on Real-Time Systems*, 2004, pp. 239–248.

[14] S. Samii, S. Rafiliu, P. Eles, and Z. Peng, "A Simulation Methodology for Worst-Case Response Time Estimation of Distributed Real-Time Systems", in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2008, pp. 556–561.

[15] R. Racu and R. Ernst, "Scheduling Anomaly Detection and Optimization for Distributed Systems with Preemptive Task-Sets", in *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, 2006, pp. 325–334.

[16] V. M. Kureychik, *Geneticheskiye algoritmy i ih primeneniye*. Taganrog: Izd-vo TRTU, 2002, 244 pp.

[17] A. B. Glonina, "Programmnoye sredstvo modelirovaniya modulnyh vychislitelnih sistem dlya proverki dopustimosti ih konfiguratsiy", *Programmniye produkty i sistemy*, vol. 30, no. 4, pp. 574–582, 2017.

MODELING AND ANALYSIS OF INFORMATION SYSTEMS, VOL. 27, NO. 2, 2020
journal homepage: www.mais-journal.ru

DISCRETE MATHEMATICS IN RELATION TO COMPUTER SCIENCE

On the Approximation of the Resource Equivalences in Petri Nets with the Invisible Transitions

V. A. Bashkin[1]

DOI: 10.18255/1818-1015-2020-2-234-253

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

MSC2020: 68Q85
Research article
Full text in English

Received April 27, 2020
After revision May 18, 2020
Accepted May 20, 2020

Two resources (submarkings) are called similar if in any marking any one of them can be replaced by another one without affecting the observable behavior of the net (regarding marking bisimulation). It is known that resource similarity is undecidable for general labelled Petri nets. In this paper we study the properties of the resource similarity and resource bisimulation (a subset of complete similarity relation closed under transition firing) in Petri nets with invisible transitions (where some transitions may be labelled with an invisible label (τ) that makes their firings unobservable for an external observer). It is shown that for a proper subclass (p-saturated nets) the resource bisimulation can be effectively checked. For a general class of Petri net with invisible transitions it is possible to construct a sequence of so-called (n, m)-equivalences approximating the largest τ-bisimulation of resources.

Keywords: resource, equivalence, Petri nets, invisible transitions, approximation.

INFORMATION ABOUT THE AUTHORS

Vladimir A. Bashkin
correspondence author

orcid.org/0000-0002-2534-1026. E-mail: v_bashkin@mail.ru
associate professor, doctor of science.

Funding: RFBR, project No 17-07-00823.

For citation: V. A. Bashkin, "On the Approximation of the Resource Equivalences in Petri Nets with the Invisible Transitions", *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 234-253, 2020.

МОДЕЛИРОВАНИЕ И АНАЛИЗ ИНФОРМАЦИОННЫХ СИСТЕМ, ТОМ 27, № 2, 2020
сайт журнала: www.mais-journal.ru

DISCRETE MATHEMATICS IN RELATION TO COMPUTER SCIENCE

Аппроксимация ресурсных эквивалентностей в сетях Петри с невидимыми переходами

В. А. Башкин[1]

DOI: 10.18255/1818-1015-2020-2-234-253

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

УДК 519.7
Научная статья
Полный текст на английском языке

Получена 27 апреля 2020 г.
После доработки 18 мая 2020 г.
Принята к публикации 20 мая 2020 г.

Два ресурса (подразметки) называются подобными, если в любой разметке любой из них может быть заменен другим, и при этом наблюдаемое поведение сети не изменится (относительно бисимуляции разметок). Известно, что подобие ресурсов неразрешимо для обыкновенных сетей Петри. В этой статье мы изучаем свойства подобия ресурсов и бисимуляции ресурсов (подмножество отношения подобия, замкнутое по срабатыванию переходов) в сетях Петри с невидимыми переходами (где некоторые переходы могут быть помечены специальной меткой (τ), что делает их срабатывания невидимыми для внешнего наблюдателя). Показано, что для собственного подкласса (p-насыщенных сетей) бисимуляция ресурсов может быть эффективно проверена. Для общего класса сетей Петри с невидимыми переходами можно построить последовательность так называемых (n, m)-эквивалентностей, аппроксимирующую наибольшую τ-бисимуляцию ресурсов.

Ключевые слова: ресурс, эквивалентность, сети Петри, невидимые переходы, аппроксимация.

ИНФОРМАЦИЯ ОБ АВТОРАХ

Владимир Анатольевич Башкин
автор для корреспонденции

orcid.org/0000-0002-2534-1026. E-mail: v_bashkin@mail.ru
доцент, д.ф.-м.н..

Финансирование: РФФИ, проект № 17-07-00823.

Для цитирования: V. A. Bashkin, “On the Approximation of the Resource Equivalences in Petri Nets with the Invisible Transitions”, *Modeling and analysis of information systems*, vol. 27, no. 2, pp. 234-253, 2020.

1. Introduction

In this paper the behavior of Petri nets is investigated from the standpoint of bisimulation equivalence. The fundamental notion of bisimulation was introduced by R. Milner [1] and D. Park [2]. Two markings of a Petri net are called bisimilar if the choice of each of them as an initial marking gives the same visible behavior of the net. In [3] P. Jančar proved that bisimulation equivalence of markings is undecidable for a general Petri net.

In [4] C. Autant et al. introduced a notion of place bisimulation — a decidable bisimulation-induced equivalence on the finite set of places, that allows to find out some non-trivial behavior-preserving net reductions. This relation and its applications were studied in [4–6].

The notion of resource similarity was introduced in [7]. In general a resource is a submarking. Two resources are similar if, having replaced one resource in any marking by another, we obtain the same observed behavior of the net. Resource bisimulation is a particular case of similarity that is closed under transition firing. Place bisimulation is a proper subset of resource bisimulation. Note that, unlike the place bisimulation [4], resource similarity and bisimulation are defined on the infinite set (of resources/submarkings).

Resource similarity and its modifications where studied in [7–9]. In particular it was proven that resource similarity is undecidable. However, it was shown that resource bisimulation can be effectively approximated and used as a basis of net reductions and adaptive control. For an overview, see [10].

This article is an extended version of the workshop report [11]. We consider an important generalization of labelled Petri nets, where some transitions may be labelled with an invisible label (tau), that makes their firings unobservable for an external observer. Quite often when analyzing the system there is a need to abstract from the excessive information about its behavior. For example, it is convenient to hide all transitions, corresponding to the internal actions of the system. The information obtained in this case can be useful, in particular, to detect additional properties of the system in terms of its interaction with the environment.

Place bisimulations in Petri nets with invisible transitions were studied by C. Autant et al. in [5]. It was shown that unlabelled sequences of steps significantly complicate the calculations. However, there are specific nontrivial subclasses of Petri nets with invisible transitions, that have some nice properties w.r.t. place bisimulation.

In this paper we apply a similar approach to the resource equivalences. It is shown that resource bisimulations can be effectively computed in some non-trivial subclasses of nets with invisible transitions.

A class of p-saturated nets is studied. In p-saturated nets the firing of any sequence of transitions with at most one visible label can be simulated by a simultaneous (independent) firing of a certain set of transitions with the same label (called parallel step). In p-saturated Petri nets τ-bisimulation coincides with the so-called τp-bisimulation [5], that takes into account parallel steps instead of transition sequences.

It is shown that in the class of p-saturated nets the weak transfer property of resource τp-bisimulation can be effectively checked. Moreover, we can underapproximate the largest τp-bisimulation by a parameterized algorithm.

It is shown that for a general class of Petri net with invisible transitions it is possible to construct a sequence of so-called (n, m)-equivalences, approximating the largest τ-bisimulation of resources.

The paper is organized as follows. Section 2 contains basic definitions. Specifically, in Subsection 2.1 we give some technical notions and lemmata on the properties of additively-transitively closed relations on multisets. Subsection 2.2 contains definitions of Petri nets and bisimulations. Subsections 2.3 and 2.4 give a short review on Petri net resources and resource equivalences (similarity and bisimulation). Section 3 deals with invisible transitions. In Subsections 3.1 and 3.2 we define the τ-generalizations of resource equivalences and study their properties. It is shown that the straightforward method of bisimulation checking with a weak transfer property is not applicable here. In Section 4 we study the subclass of p-saturated nets and the corresponding notion of τp-bisimulation. In Subsection 4.3 we present an algorithm, computing

the parameterized underapproximation of largest τp-bisimulation. Section 5 is devoted to the general case of Petri nets with invisible transitions. A parameterized approximation procedure for resource bisimulation is defined and studied. Section 6 contains some conclusions.

2. Preliminaries

2.1. Relations on multisets

Denote by ε an empty sequence. Let X and Y be two sets. Let $\sigma \in X^*$ be a sequence over X. Denote by $\sigma_{|Y}$ a *projection* of σ onto Y such that for an empty sequence $\sigma = \varepsilon$ we have $\sigma_{|Y} =_{def} \varepsilon$ and for a non-empty sequence $\sigma = a\delta$ with $a \in X$ and $\delta \in X^*$ we have $\sigma_{|Y} =_{def} a\delta_{|Y}$ for $a \in Y$ and $\sigma_{|Y} =_{def} \delta_{|Y}$ for $a \notin Y$.

A *multiset* M over a set X is a mapping $M : X \longrightarrow Nat$, where Nat is the set of natural numbers (including zero), i.e. a multiset may contain several copies of the same element.

Size of a multiset is defined as follows: $|M| = \sum_{x \in X} M(x)$. A multiset M is finite if a set $\{x \in X \mid M(x) > 0\}$ is finite. By $\mathcal{M}(X)$ we denote the set of all finite multisets over X. An empty multiset is denoted by $\emptyset$.

The operations and relations of set theory are naturally extended to finite multisets. Let $M_1, M_2, M_3 \in \mathcal{M}(X)$. Then:

- $M_1 = M_2 \iff_{def} \forall x \in X\ M_1(x) = M_2(x)$;
- $M_1 \subseteq M_2 \iff_{def} \forall x \in X\ M_1(x) \leq M_2(x)$;
- $M_1 \subset M_2 \iff_{def} M_1 \subseteq M_2 \wedge \exists x \in X\ M_1(x) \leq M_2(x)$;
- $M_1 = M_2 + M_3 \iff_{def} \forall x \in X\ M_1(x) = M_2(x) + M_3(x)$;
- $M_1 = M_2 \cap M_3 \iff_{def} \forall x \in X\ M_1(x) = min\{M_2(x), M_3(x)\}$;
- $M_1 = M_2 - M_3 \iff_{def} \forall x \in X\ M_1(x) = max\{0, M_2(x) - M_3(x)\}$;
- $M_1 = kM_2,\ k \in Nat \iff_{def} \forall x \in X\ M_1(x) = kM_2(x)$;
- $M_1 = (M_2)_{|Y},\ Y \subseteq X \iff_{def} \forall x \in X\ M_1(x) = M_2(x)$ for $x \in Y$ and $M_1(x) = 0$ otherwise.

Non-negative integer vectors are often used to encode multisets. Actually, the set of all multisets over finite X is a homomorphic image of $Nat^{|X|}$.

A binary relation $B \subseteq Nat^k \times Nat^k$ is a congruence if it is an equivalence relation and whenever $(v, w) \in B$ then $(v + u, w + u) \in B$ (here '+' denotes coordinate-wise addition).[1] It was proven by L. Redei [12] that every congruence on Nat^k is generated by a finite set of pairs. Later P. Jančar [3] and J. Hirshfeld [13] presented a shorter proof and also showed that every congruence on Nat^k is a semilinear relation, i.e. it is a finite union of linear sets.

Let B^{AT} denote the additive-transitive closure (AT-closure) of the relation $B \subseteq \mathcal{M}(X) \times \mathcal{M}(X)$ (the minimal congruence, containing B).

Let $B \subseteq \mathcal{M}(X) \times \mathcal{M}(X)$ be a binary relation on multisets. A relation B' is called an *AT-basis* of B iff $(B')^{AT} = B^{AT}$. An AT-basis B' is called *minimal* iff there is no $B'' \subset B'$ such that $(B'')^{AT} = B^{AT}$.

Now we construct a special kind of minimal AT-basis for B. Define a partial order $\sqsubseteq$ on the set $B \subseteq \mathcal{M}(X) \times \mathcal{M}(X)$ of pairs of multisets as follows:

1. For loop (i.e. reflexive) pairs let
$$(r_1, r_1) \sqsubseteq (r_2, r_2) \overset{def}{\iff} r_1 \subseteq r_2;$$
2. For two non-loop pairs, the maximal loop constituents and the addend pairs of nonintersecting multisets are compared separately
$$(r_1 + o_1, r_1 + o_1') \sqsubseteq (r_2 + o_2, r_2 + o_2') \overset{def}{\iff}$$
$$\overset{def}{\iff} o_1 \cap o_1' = \emptyset\ \&\ o_2 \cap o_2' = \emptyset\ \&\ r_1 \subseteq r_2\ \&\ o_1 \subseteq o_2\ \&\ o_1' \subseteq o_2'.$$

[1] Note that it can be easily seen that if B is a congruence and $(v, w), (u, x) \in B$ then also $(v + u, w + x) \in B$.

3. a loop pair and a non-loop pair are always incomparable.

Let B_s denote the set of all minimal (with respect to $\sqsubseteq$) elements of B^{AT}.

Theorem 1. *[8] Let $B \subseteq \mathcal{M}(X) \times \mathcal{M}(X)$ be a symmetric and reflexive relation. Then B_s is an AT-basis of B and B_s is finite.*

We call B_s the *ground basis* of B. Obviously, it is finite.

There is also a useful

Lemma 1. *[8] Let $B \subseteq \mathcal{M}(X) \times \mathcal{M}(X)$ be a symmetric and reflexive relation, $(r, s) \in B^{AT}$. Then there exists a finite chain of pairs*

$$(r, a_1), (a_1, a_2), \dots, (a_{k-1}, a_k), (a_k, s) \in (B_s)^A,$$

where $(B_s)^A$ is the additive closure of B_s.

2.2. Labelled Petri nets and bisimulations

Let P and T be disjoint sets of *places* and *transitions* and let $F : (P \times T) \cup (T \times P) \longrightarrow Nat$. Then $N = (P, T, F)$ is a *Petri net*. a *marking* in a Petri net is a function $M : P \longrightarrow Nat$, mapping each place to some natural number (possibly zero). Thus a marking may be considered as a multiset over the set of places. Pictorially, P-elements are represented by circles, T-elements by boxes, and the flow relation F by directed arcs. Places may carry tokens represented by filled circles. a current marking M is designated by putting $M(p)$ tokens into each place $p \in P$. Tokens residing in a place are often interpreted as resources of some type consumed or produced by a transition firing. a marked Petri net (N, M_0) is a Petri net N together with a given initial marking M_0.

For a transition $t \in T$ the *preset* ${}^\bullet t$ and the *postset* $t^\bullet$ are defined as the multisets over P such that ${}^\bullet t(p) = F(p, t)$ and $t^\bullet(p) = F(t, p)$ for each $p \in P$.

A transition $t \in T$ is *enabled* in a marking M iff ${}^\bullet(t) \subseteq M$. An enabled transition t may *fire* yielding a new marking $M' =_{\text{def}} M - {}^\bullet t + t^\bullet$, i.e. $M'(p) = M(p) - F(p, t) + F(t, p)$ for each $p \in P$ (denoted $M \xrightarrow{t} M'$).

Let $\sigma \in T^*$ be a sequence of transition (possibly empty), $t \in T$ – a transition. The pre- and postcondition for a non-empty sequence are defined inductively:

$${}^\bullet(t\sigma) =_{def} {}^\bullet t + ({}^\bullet\sigma - t^\bullet), \qquad (\sigma t)^\bullet =_{def} t^\bullet + (\sigma^\bullet - {}^\bullet t).$$

A sequence $\sigma \in T^*$ is *enabled* in M iff ${}^\bullet\sigma \subseteq M$. An enabled sequence may *fire* yielding a new marking $M' =_{\text{def}} M - {}^\bullet\sigma + \sigma^\bullet$ (denoted $M \xrightarrow{\sigma} M'$).

A multiset of transitions may *fire in parallel* (concurrently), if there are enough tokens for all of them. a transition may fire in parallel with itself. The concurrent firing of a multiset of transitions is called *a parallel step*. The pre- and postcondition for a multiset of transitions $U \in \mathcal{M}(T)$ are:

$${}^\bullet U =_{def} \sum_{t \in T} U(t) \times {}^\bullet t, \qquad U^\bullet =_{def} \sum_{t \in T} U(t) \times t^\bullet.$$

A parallel step $U \in \mathcal{M}(T)$ is *enabled* in M iff ${}^\bullet U \subseteq M$. An enabled parallel step may *fire* yielding a new marking $M' =_{\text{def}} M - {}^\bullet U + U^\bullet$ (denoted $M \xrightarrow{U} M'$).

Obviously, we have ${}^\bullet(U + W) = {}^\bullet U + {}^\bullet W, (U + W)^\bullet = U^\bullet + W^\bullet$.

To observe the net behavior transitions are labelled by special labels representing observable actions or events. Let *Act* be a set of action names. A *labelled Petri net* is a tuple $N = (P, T, F, l)$, where (P, T, F) is a Petri net and $l : T \longrightarrow Act$ is a labelling function. It can be generalized to non-empty sequences:

$$\text{for } \alpha \in T^* \text{ s.t. } \alpha = t\beta \text{ with } t \in T \text{ and } \beta \in T^* \text{ we have } l(\alpha) =_{def} l(t)l(\beta).$$

And also to multisets of transitions (note that in this case labels are not sequences but multisets of action names):

$$\text{for } U \in \mathcal{M}(T) \qquad l(U) =_{def} \sum_{t \in T} U(t) \times l(t).$$

Let $N = (P, T, F, l)$ be a labelled Petri net. We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the *transfer property* iff for all $(M_1, M_2) \in B$ and for every step $t \in T$, s.t. $M_1 \xrightarrow{t} M_1'$, there exists an imitating step $u \in T$, s.t. $l(t) = l(u)$, $M_2 \xrightarrow{u} M_2'$ and $(M_1', M_2') \in B$.

A relation B is called a *marking bisimulation*, if both B and B^{-1} conform to the transfer property.

It is known that a union of two marking bisimulations is a marking bisimulation. Hence for every labelled Petri net there exists the largest marking bisimulation (a union of all bisimulations; denoted by ~) and this bisimulation is an equivalence. It was proved by P. Jančar [3], that the marking bisimulation is undecidable for Petri nets. More precisely, it is undecidable whether two markings (of the same net) are marking bisimilar, even if restricted to nets with only two unbounded places.

2.3. Resource similarity

Informally, resources are parts of markings which may or may not provide some particular kind of observable net behavior.

Definition 1. *[8] Let $N = (P, T, F, l)$ be a labelled Petri net. a* resource $R \in \mathcal{M}(P)$ *in a Petri net N is a multiset over the set of places P.*

Resources r and s in N are called similar *(denoted $r \approx s$) iff for every marking $R \in \mathcal{M}(P)$, $r \subseteq R$ implies $R \sim R - r + s$.*

Thus if two resources are similar, then in every marking each of these resources can be replaced by the other without changing the observable behavior of the system. Here we consider the observability modulo action names: the external observer can see events (labels of fired transitions) but cannot distinguish local states (tokens). Some examples of similar resources are shown in Fig. 1.

Figure a) shows a Petri net containing two transitions labeled with the same label a and leading to the same marking p_3. Here the resources p_1 and p_2 are similar, as they lead to a completely identical observable behavior — action a producing a single token in p_3. Moreover, all the resources containing the same number of tokens in p_1 and p_2 are similar.

Figure b) shows a simple net consisting of a single transition. In this case the resource p_2 is similar to an empty resource, since it does not affect the behavior of the net (the place p_2 is redundant).

Figure c) depicts a cycle consisting of one transition and one place. Note that the set of markings of this net can be divided into two disjoint subsets — empty marking and all the others. With empty marking, the transition can not fire, for all others — it can fire any number of times. Note that for this net the largest marking bisimulation and the resource similarity coincide. Also note that marking bisimulation takes into account only steps made of single transitions hence no auto-concurrency can be considered here.

Figure d) shows a more complex situation. We have $p_1 \approx p_2 + p_3$, that is, replacing one token in p_1 by two tokens (one in p_2 and one in p_3) does not affect the observable behavior of the net as a whole.

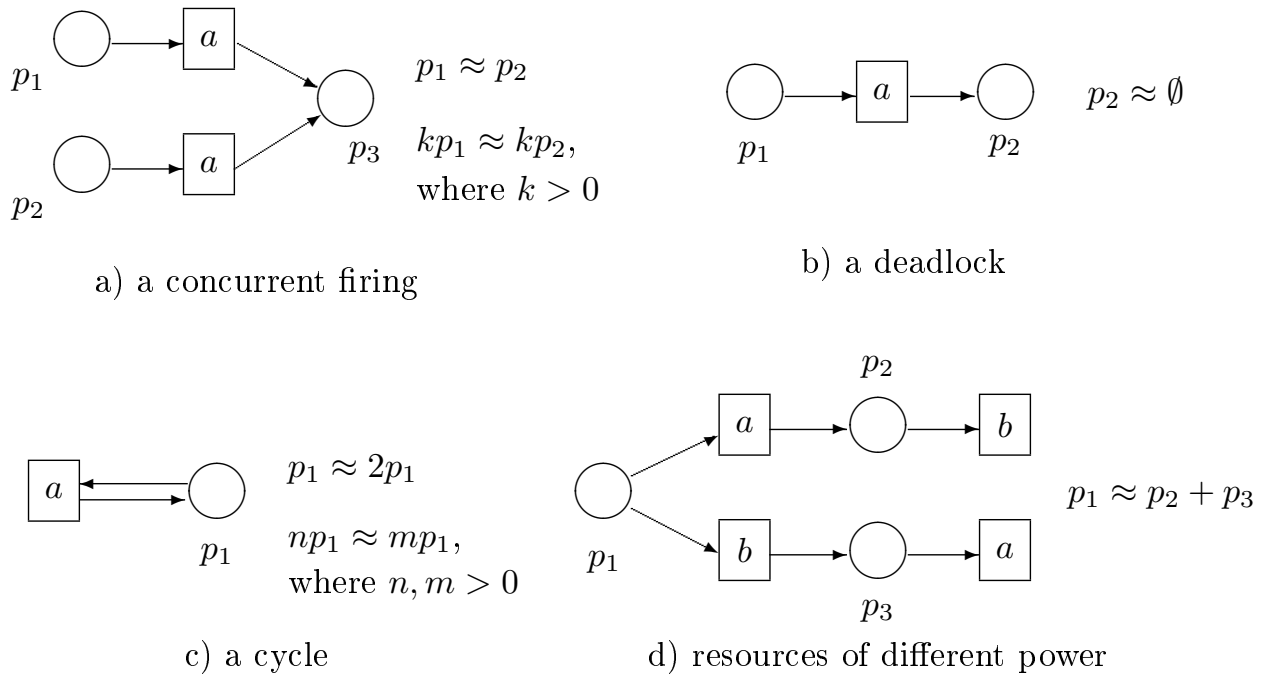


Fig. 1. Examples of similar resources

Рис. 1. Пример подобных ресурсов

The similarity relation is an equivalence [8]. Moreover, it is a congruence w.r.t. multiset addition:

Proposition 1. *[8] Let $N = (P, T, F, l)$ be a labelled Petri net, let r, s, u, v be resources of the net N. Then $r \approx s \,\&\, u \approx v \implies r + u \approx s + v$.*

Hence it has a finite ground basis. Unfortunately, from the undecidability of a stronger relation of place fusion [6] we get

Theorem 2. *[8] The resource similarity is undecidable for labelled Petri nets.*

2.4. Resource bisimulation

The resource similarity is quite fundamental, but the undecidability makes it not very useful in practice. So we studied a number of other non-trivial finitely-based resource equivalence relations, retaining the observable system's behavior. The most interesting of them is a resource bisimulation:

Definition 2. *[8] Let $N = (P, T, F, l)$ be a labelled Petri net. An equivalence relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is called a resource bisimulation if B^{AT} is a marking bisimulation.*

Note that an AT-closure of a resource similarity relation is not necessarily a marking bisimulation (it is still an open question [10]). However, we already know that each resource bisimulation B is a subset of resource similarity relation ($\approx$). The following theorem states this and some other important properties of resource bisimulations.

Theorem 3. *[8] Let $N = (P, T, F, l)$ be a labelled Petri net. Then*

1. *if $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is a resource bisimulation and $(r_1, r_2) \in B$ then $r_1 \approx r_2$;*
2. *if B_1, B_2 are resource bisimulations for N then $B_1 \cup B_2$ is a resource bisimulation for N;*
3. *for any N there exists the largest resource bisimulation (denoted by $B(N)$), and it is an equivalence.*

Therefore $B(N)$ (as well as any other resource bisimulation) also has a finite ground basis.

The AT-closure of a resource bisimulation is a marking bisimulation, and hence, it conforms to the transfer property. Resource bisimulations satisfy a weak variant of the transfer property, considering only minimal pairs of markings that contain the corresponding resources and enable the corresponding transitions.

We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to *the weak transfer property* if for all $(r, s) \in B$, for each $t \in T$, such that ${}^\bullet t \cap r \neq \emptyset$, there exists an imitating transition $u \in T$, such that $l(t) = l(u)$ and, writing M_1 for ${}^\bullet t \cup r$ and M_2 for ${}^\bullet t - r + s$, we have $M_1 \xrightarrow{t} M_1{}'$ and $M_2 \xrightarrow{u} M_2{}'$ with $(M_1', M_2') \in B^{AT}$.

Theorem 4. *[8] Let $N = (P, T, F, l)$ be a labelled Petri net. A relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is a resource bisimulation iff B is an equivalence relation and it conforms to the weak transfer property.*

Due to this theorem to check whether a given finite relation B is a resource bisimulation, one needs to verify the weak transfer property for only a finite number of pairs of resources. In [8] we have shown that the largest resource bisimulation for resources with a bounded number of tokens can be effectively constructed (more precisely, it requires $O(\max\{|P|\,\mathcal{R}^9, |T|^2|P|\,\mathcal{R}^7\})$ steps, where $\mathcal{R}$ is the number of resources in the consideration).

3. Petri nets with invisible transitions

In this section we investigate the possibilities of effectively constructing bisimulation-preserving relations for an extended class of systems — Petri nets with invisible transitions.

To distinguish visible and invisible transitions, a special τ symbol is added to the set of labels: $Act_\tau = Act \cup \{\tau\}$.

Definition 3. *A* labelled Petri net with invisible transitions *is a tuple $N = (P, T, F, l)$, where (P, T, F) is a Petri net and $l : T \longrightarrow Act_\tau$ is an extended labelling function.*

Let $\sigma, \sigma' \in (Act_\tau)^*$ be sequences of action labels. Denote $\sigma =_\tau \sigma' \iff_{def} \sigma_{|Act} = \sigma'_{|Act}$ ("equal modulo τ"). For example, "$\tau\tau a\tau$" $=_\tau$ "a".

Similarly, let $U, U' \in \mathcal{M}(Act_\tau)$ be multisets of action labels. Denote $U =_\tau U' \iff_{def} U_{|Act} = U'_{|Act}$. For example, $\{a, \tau, a, b, \tau\} =_\tau \{a, a, b\}$.

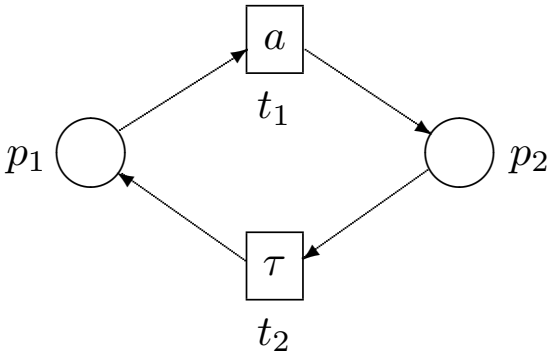
3.1. τ-bisimulation

Let $N = (P, T, F, l)$ be a labelled Petri net with invisible transitions. We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the *τ-transfer property* iff for all $(M_1, M_2) \in B$ and for every step $t \in T$, s.t. $M_1 \xrightarrow{t} M_1'$, there exists an imitating sequence of steps $\sigma \in T^*$ s.t. $l(t) =_\tau l(\sigma)$, $M_2 \xrightarrow{\sigma} M_2'$ and $(M_1', M_2') \in B$.

A relation B is called a *marking τ-bisimulation*, if both B and B^{-1} conform to the τ-transfer property. The largest τ-bisimulation is denoted by $\sim_\tau$.

Marking bisimulation is a special case of marking τ-bisimulation (for nets with no τ-s). It is a stronger relation. Consider as an example the net depicted in Fig. 2. Markings p_1 and p_2 are not bisimilar, because at p_2 no transition with label a is active. But they are τ-bisimilar, because the invisible firing of t_2 changes the marking from p_2 to p_1.

In particular, this implies the undecidability of marking τ-bisimulation in Petri nets with invisible transitions [3].



$p_1 \not\sim p_2$

$p_1 \sim_\tau p_2$

Fig. 2. τ-bisimulation is weaker than bisimulation

Рис. 2. τ-бисимуляция слабее, чем обычная бисимуляция

3.2. Resource similarity and bisimulation

The definition of resource similarity can be naturally generalized to the case of nets with invisible transitions:

Definition 4. *Let $N = (P, T, F, l)$ be a labelled Petri net with invisible transitions. Resources r and s are called* τ-similar *(denoted $r \approx_\tau s$) iff for every marking R, $r \subseteq R$ implies $R \sim_\tau R - r + s$.*

We can show that resource τ-similarity has all basic properties of resource similarity:

Proposition 2.
1. *Resource τ-similarity is closed under addition and is transitive; hence it has finite AT-basis.*
2. *Resource τ-similarity is undecidable.*

Proof. 1) From the definitions.

2) From Th. 2 (note that τ-similarity is a generalization of basic resource similarity). □

The definition of resource bisimulation also can be easily generalized:

Definition 5. *Let $N = (P, T, F, l)$ be a labelled Petri net with invisible transitions. An equivalence relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is called a* resource τ-bisimulation *if B^{AT} is a marking τ-bisimulation.*

Proposition 3. *Let $N = (P, T, F, l)$ be a labelled Petri net with invisible transitions. Then*

1. if $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is a resource τ-bisimulation and $(r_1, r_2) \in B$ then $r_1 \approx_\tau r_2$;
2. if $B_1, B_2 \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ are resource τ-bisimulations then $B_1 \cup B_2$ is a resource τ-bisimulation;
3. for any N there exists the largest resource τ-bisimulation (denoted by $B_\tau(N)$), and it is an equivalence.

Proof. 1) We need to prove that $r_1 \approx_\tau r_2$: for any $R \in \mathcal{M}(P)$ s.t. $r_1 \subseteq R$ we have $R \sim_\tau R - r_1 + r_2$.

Denote $r' = R - r_1$. The pair $(R, R - r_1 + r_2)$ can be represented as $(r_1 + r', r_2 + r')$, therefore it belongs to B^{AT}. Since B is a resource τ-bisimulation, B^{AT} is a marking τ-bisimulation, and hence it is a subset of a largest marking τ-bisimulation ($\sim_\tau$). So, we obtained $R \sim_\tau R - r_1 + r_2$.

2) Denote $B = B_1 \cup B_2$. We need to prove that B is a resource τ-bisimulation: for any $(M_1, M_2) \in B^{AT}$ we have $M_1 \sim_\tau M_2$.

Consider the structure of (M_1, M_2). From Lm. 1 we have

$$(M_1, a_1), (a_1, a_2), \dots, (a_{k-1}, a_k), (a_k, M_2) \in (B_s)^A$$

for some finite k, where $(B_s)^A$ is the additive closure of B_s.

It can be easily seen that $B_s \subseteq (B_1)_s \cup (B_2)_s$, hence for any $(X, Y) \in (B_s)^A$ we have $X = X_1 + X_2$, $Y = Y_1 + Y_2$ s.t. $(X_1, Y_1) \in ((B_1)_s)^A$ and $(X_2, Y_2) \in ((B_2)_s)^A$.

From the reflexivity of B_1 and B_2 and additive closureness of $(B_1)^{AT}$ and $(B_2)^{AT}$ we have $(X_1 + X_2, Y_1 + X_2) \in (B_1)^{AT}$ and $(Y_1 + X_2, Y_1 + Y_2) \in (B_2)^{AT}$. Both B_1 and B_2 are resource τ-bisimulations, so $(B_1)^{AT}$ and $(B_2)^{AT}$ are marking τ-bisimulations. Therefore they are both contained in the largest τ-bisimulation ($\sim_\tau$), so we have $X_1 + X_2 \sim_\tau Y_1 + X_2$ and $Y_1 + X_2 \sim_\tau Y_1 + Y_2$. The bisimulation is transitive, hence $X_1 + X_2 \sim_\tau Y_1 + Y_2$.

So for any $(X, Y) \in (B_s)^A$ we have $X \sim_\tau Y$. Applying this reasoning to the pairs in our chain, we obtain $M_1 \sim_\tau a_1, a_1 \sim_\tau a_2, \dots, a_{k-1} \sim_\tau a_k, a_k \sim_\tau M_2$. Hence, $M_1 \sim_\tau M_2$.

3) The third statement is an immediate corollary of the second one. The largest resource τ-bisimulation can be constructed as the union of all resource τ-bisimulations for N. □

Definition 6. *We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the* weak τ-transfer property *if for all $(r, s) \in B$, $t \in T$ s.t. ${}^\bullet t \cap r \neq \emptyset$, there exists an imitating sequence of transitions $\sigma \in T^*$ s.t. $l(t) =_\tau l(\sigma)$ and, denoting $M_1 = {}^\bullet t \cup r$ and $M_2 = {}^\bullet t - r + s$, we have $M_1 \xrightarrow{t} M_1'$ and $M_2 \xrightarrow{\sigma} M_2'$ with $(M_1', M_2') \in B^{AT}$.*

Th. 4 in the case of Petri nets with invisible transitions works only in one direction:

Proposition 4. *If the relation conforms to the τ-transfer property then it conforms to the weak τ-transfer property; there exist relations, conforming to the weak τ-transfer property and not conforming to the τ-transfer property.*

Proof. ($\Longrightarrow$) Since the weak τ-transfer property is the τ-transfer property for a bounded (finite) subset of pairs of resources.

($\Longleftarrow$) Consider the net depicted in Fig. 3 (this example is taken from [5]) and a relation

$$B = Id(P) \cup \{(p_1, p_2), (p_2, p_1), (p_3, p_4), (p_4, p_3)\},$$

where $Id(P)$ is an identity relation s.t. $\forall x, y \in P \quad (x, y) \in Id(P) \Longleftrightarrow x = y$.

B conforms to the weak τ-transfer property. At the same time B is not a resource τ-bisimulation. Consider markings $M_1 = p_1 + p_3$ and $M_2 = p_2 + p_4$. The pair (M_1, M_2) belongs to the relation B^{AT}, but the markings are not bisimilar, because an action a is possible at M_2 (transition t_3) and is impossible at M_1. □

Hence the weak τ-transfer property can not be used to construct bisimulation. In the case of systems with invisible transitions it is even more important to strengthen the considered relations and/or to restrict the considered class of Petri nets.

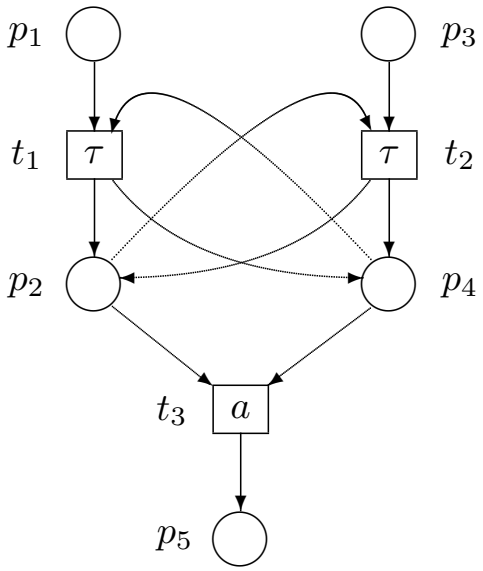


Fig. 3. Th. 4 does not hold for Petri nets with invisible transitions

Рис. 3. Теорема 4 не выполняется для сетей Петри с невидимыми переходами

4. Underapproximations of τ-similarity in saturated nets

4.1. Saturated nets

There exists a wide and important subclass of Petri nets with invisible transitions for which resource τ-bisimulation can be constructed using weak transfer property — so-called "p-saturated nets". In p-saturated nets [5] the firing of any sequence of transitions with at most one visible label can be simulated by a simultaneous (independent) firing of a certain set of transitions with the same label (called "parallel step").

Denote the set of non-empty transition sequences with at most one visible label:

$$T^{\times} =_{def} \{\sigma \in T^{*} \mid l(\sigma) \in Act_{\tau}\}.$$

Definition 7. *A labelled Petri net with invisible transitions $N = (P, T, F, l)$ is called* p-saturated *(or simply* saturated*), if for any sequence of transitions $\sigma \in T^{\times}$ there exists a parallel step $U \in \mathcal{M}(T)$ s.t. ${}^{\bullet}U = {}^{\bullet}\sigma$, $U^{\bullet} = \sigma^{\bullet}$ and, denoting by U_{σ} the multiset of transitions, participating in σ, we have $l(U) =_{\tau} l(U_{\sigma})$.*

In addition to saturated nets, there is an even broader class of *saturable* Petri nets. These are nets that can be transformed into saturated by adding a finite number of transitions while preserving the behavior of the net (in the sense of τ-bisimilarity). In Fig. 4 a saturated net is shown, obtained by adding the transition t_3 to the unsaturated net.

It is known [5] that a net is p-saturated iff it is $2p$-saturated, i.e. all sequences of length 2 are saturated by parallel steps.

Not all nets are saturable [5]. An example is given in Fig. 5. Here all transition sequences has the same precondition (a single token in the upper place) and different postconditions. So there is an infinite set of different transition sequences with different postconditions. On the other hand, the structure of the net also implies that all possible parallel steps with the same precondition (a single token in the upper place) would necessarily contain a single transition. Hence the number of different imitating parallel steps is always finite and equal to the number of existing transition. The saturation would not help, because it can not introduce an infinite number of new transitions.

It is also easy to see that the net is saturable iff its "invisible subnet" is saturable (an invisible subnet is a net, obtained by removing all visible transitions).

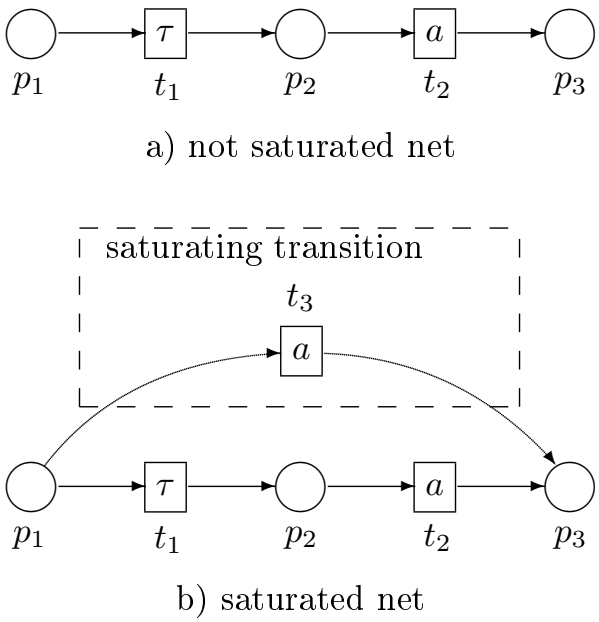


Fig. 4. An example of net saturation

Рис. 4. Пример насыщения сети

4.2. τp-bisimulation

In [5] an equivalence stronger than τ-bisimulation was defined, called τp-bisimulation of markings. The transition in this case is modeled not by a sequence of transitions, but by a parallel step.

Definition 8. *[5] Let $N = (P, T, F, l)$ be a labelled Petri net with invisible transitions. We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the* τp-transfer property *if for all $(M_1, M_2) \in B$ and for each $t \in T$ s.t. $M_1 \xrightarrow{t} M_1'$, there exists an imitating parallel step $U \in \mathcal{M}(T)$ s.t. $\{l(t)\} =_\tau l(U)$, $M_2 \xrightarrow{U} M_2'$ and $(M_1', M_2') \in B$.*

Definition 9. *[5] A relation B is called a* marking τp-bisimulation, *if both B and B^{-1} conform to the τp-transfer property.*

It is known [5] that for any net there exists the largest τp-bisimulation (denoted by $\sim_{\tau p}$).
In saturated Petri nets τp-bisimulation coincides with τ-bisimulation [5]:

$$M_1 \sim_{\tau p} M_2 \quad \Leftrightarrow \quad M_1 \sim_\tau M_2.$$

Now we are ready to define a resource τp-similarity:

Definition 10. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. Resources r and s are called* τp-similar *(denoted $r \approx_{\tau p} s$) iff for every marking R, $r \subseteq R$ implies $R \sim_{\tau p} R - r + s$.*

From the equality of $\sim_{\tau p}$ and $\sim_\tau$ in saturated nets we immediately have:

Corollary 1. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions, $r, s \in \mathcal{M}(P)$. Then*

$$r \approx_{\tau p} s \quad \Leftrightarrow \quad r \approx_\tau s.$$

So, in saturated nets it is sufficient to look for τp-similarities.

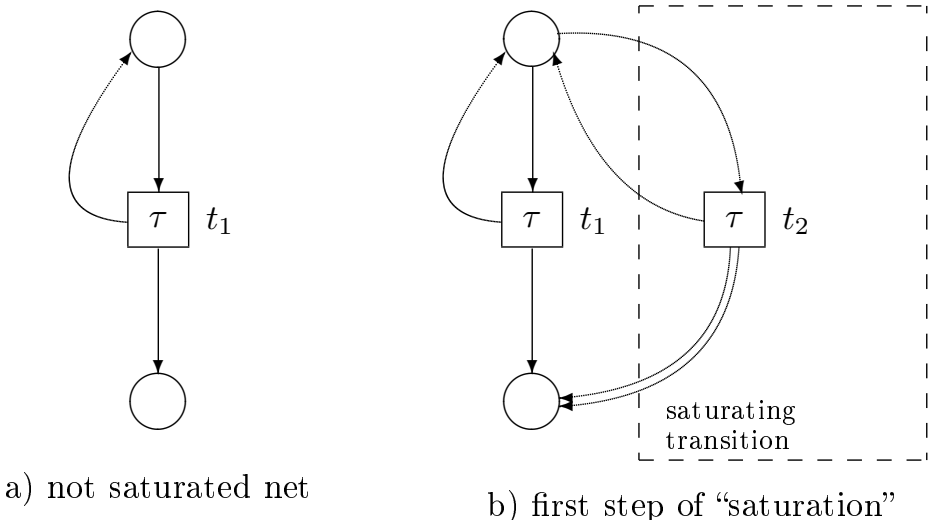


Fig. 5. Not saturable net

Рис. 5. Не насыщаемая сеть

Definition 11. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. An equivalence relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is called a* resource τp-bisimulation *if B^{AT} is a marking τp-bisimulation.*

In the case of τp-relations all basic properties also hold:

Proposition 5. *1. Resource τp-similarity is closed under addition and transitivity; so it has finite AT-basis.*
2. Resource τp-similarity is undecidable.
3. If $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is a resource τp-bisimulation and $(r_1, r_2) \in B$ then $r_1 \approx_{\tau p} r_2$.
4. If $B_1, B_2 \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ are resource τp-bisimulations then $B_1 \cup B_2$ is a resource τp-bisimulation;
5. For any N there exists the largest resource τp-bisimulation (denoted by $B_{\tau p}(N)$), and it is an equivalence.

Proof. 1) Immediately from the definition of resource τp-similarity.
2) From Cor. 1 and Prop. 2.2 (the undecidability of $(\approx_\tau)$).
3) Immediately from the definitions.
4) The proof is almost the same as in Prop. 3: the only difference is that we consider not an imitating transition but an imitating parallel step.
5) Note that we can take a union of all resource τp-bisimulations. □

Definition 12. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the* weak τp-transfer property *if for all $(r, s) \in B$, $t \in T$ s.t. ${}^\bullet t \cap r \neq \emptyset$, there exists an imitating parallel step $U \in \mathcal{M}(T)$ s.t. $l(t) =_\tau l(U)$ and, denoting $M_1 = {}^\bullet t \cup r$ and $M_2 = {}^\bullet t - r + s$, we have $M_1 \xrightarrow{t} M_1'$ and $M_2 \xrightarrow{U} M_2'$ with $(M_1', M_2') \in B^{AT}$.*

In saturated nets the weak τp-transfer property is a necessary and sufficient condition for its extended version, which guarantees the imitation of a parallel step rather than a single transition:

Definition 13. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the* extended weak τp-transfer property *if for all $(r, s) \in B$ and any parallel step $V \in \mathcal{M}(T)$ s.t. ${}^\bullet V \cap r \neq \emptyset$, there exists an imitating parallel step $U \in \mathcal{M}(T)$ s.t. $l(V) =_\tau l(U)$ and, denoting $M_1 = {}^\bullet V \cup r$ and $M_2 = {}^\bullet V - r + s$, we have $M_1 \xrightarrow{V} M_1'$ and $M_2 \xrightarrow{U} M_2'$ with $(M_1', M_2') \in B^{AT}$.*

Lemma 2. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. The relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the weak τp-transfer property iff it conforms to the extended weak τp-transfer property.*

Proof. ($\Leftarrow$) Since the weak transfer property is a special case of the extended weak transfer property.

($\Rightarrow$) We need to show that for any $(M_1, M_2) \in B^{AT}$ and a parallel step $V = \{t_1, \dots, t_k\} \in \mathcal{M}(T)$ with $M_1 \xrightarrow{V} M_1'$ there exists an imitating parallel step $U \in \mathcal{M}(T)$ with the same visible label $l(V) =_\tau l(U)$ s.t. and $M_2 \xrightarrow{U} M_2'$ and $(M_1', M_2') \in B^{AT}$.

Consider the transition firing $M_1 \xrightarrow{t_1} M_1^1$. From the weak τp-transfer property it follows that this transition has an imitating parallel step $M_2 \xrightarrow{W_1} M_2^1$ such that $(M_1^1, M_2^1) \in B^{AT}$.

Note that $V = \{t_1, \dots, t_k\}$ is a parallel step at marking M_1, hence after the firing of one of these transitions all other are still enabled. Therefore we can repeat the previous reasoning for the new pair of markings $(M_1^1, M_2^1) \in B^{AT}$ and transition t_2. And continue this until t_k:

$$
\begin{array}{ccc}
M_1 & B^{AT} & M_2 \\
t_1 \downarrow & & \downarrow W_1 \\
M_1^1 & B^{AT} & M_2^1 \\
t_2 \downarrow & & \downarrow W_2 \\
\dots & & \dots \\
t_k \downarrow & & \downarrow W_k \\
M_1' = M_1^k & B^{AT} & M_2^k = M_2'
\end{array}
$$

At the end we got a sequence of parallel steps

$$M_2 \xrightarrow{W_1} M_2^1 \xrightarrow{W_2} M_2^2 \xrightarrow{W_3} \dots \xrightarrow{W_k} M_2^k = M_2',$$

imitating the firing of parallel step $M_1 \xrightarrow{V} M_1'$. The net is saturated so for any sequence of transitions (note that a parallel step also can be considered as a sequence of transitions) there exists an imitating parallel step U with the same label, precondition and postcondition ($M_2 \xrightarrow{U} M_2'$). □

Note that, unlike the weak transfer property, the extended weak transfer property can not be effectively checked by the search of resource pairs, since the set of parallel steps is infinite.

Theorem 5. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. An equivalence relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the weak τp-transfer property iff B is a resource τp-bisimulation.*

Proof. ($\Leftarrow$) Since the weak τp-transfer property is the τp-transfer property for a bounded (finite) subset of pairs of resources.

($\Rightarrow$) The proof is similar to the proof of Th. 4, with the additional use of Lm. 2. We need to show that B^{AT} conform to the τp-transfer property, i.e. for any $(M_1, M_2) \in B^{AT}$ and $t \in T$ with $M_1 \xrightarrow{t} M_1'$ there exists an imitating parallel step $U \in \mathcal{M}(T)$ with $l(t) = l(U)$, $M_2 \xrightarrow{U} M_2'$ and $(M_1', M_2') \in B^{AT}$.

Consider a pair of markings $(M_1, M_2) \in B^{AT}$. From Lm. 1 this pair can be obtained by a transitive closure of several pairs from B^A (additive closure of B):

$$(H_1, H_2), (H_2, H_3), \dots, (H_{k-1}, H_k) \in B^A, \text{ where } H_1 = M_1, H_k = M_2.$$

Consider the pair (H_1, H_2).

$$(H_1, H_2) = (r_1 + r_2 + \dots + r_l, s_1 + s_2 + \dots + s_l), \text{ where } (r_i, s_i) \in B$$

$H_1 = {}^\bullet t \cup r_1 + F_1$. Due to the weak transfer property for the pair (r_1, s_1) there exists an imitating parallel step $V \in \mathcal{M}(T)$ s.t. $l(t) = l(V)$, ${}^\bullet t \cup r_1 \xrightarrow{t} G_1$ and ${}^\bullet t - r_1 + s_1 \xrightarrow{V} G_2$, where $(G_1, G_2) \in B^{AT}$.

Since ${}^\bullet t \cup r_1 \subseteq H_1$, we can add the resource $F = H_1 - {}^\bullet t \cup r_1$ to preconditions and postconditions:

$$
\begin{gathered}
{}^\bullet t \cup r_1 + F \xrightarrow{t} G_1 + F \\
{}^\bullet t - r_1 + s_1 + F \xrightarrow{V} G_2 + F
\end{gathered}
$$

From the reflexivity of B and the additive closure of B^{AT} the new pair of markings is also decomposable by B : $(G_1 + F, G_2 + F) \in B^{AT}$.

We obtained a new marking $H_1' = {}^\bullet t - r_1 + s_1 + F = H_1 - r_1 + s_1$. Note that it still contains $r_2 + \cdots + r_l$. Therefore, we can apply the same reasoning one more time, replacing resource r_2 by the bisimilar resource s_2, now using Lm. 2 and constructing an imitating parallel step not for a transition but for a parallel step V.

Apply this $l - 1$ times. Using transitive closure of B^{AT}, at the end we obtain a parallel step W that can imitate t at marking H_2.

Now proceed to the next pair (H_2, H_3) and repeat the procedure for the parallel step W. And so on, until the last pair (H_{k-1}, H_k). Finally we obtain a parallel step U that can imitate t at marking $H_k = M_2$. □

Thus, in saturated nets the weak τp-transfer property can be used in the construction of resource τp-bisimulation.

4.3. Underapproximation

As in ordinary Petri nets (without invisible transitions), in the case of saturated (saturable) nets with invisible transitions there is a way of constructing an approximation of the maximal resource τp-bisimulation. If we consider not an infinite set of network resources, but only its finite subset, then it will be possible to check the weak τp-transfer property.

Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions, $q \in Nat$ — some parameter. By $\mathcal{M}_q(P)$ we denote the set of all resources, containing not more than q tokens in the net: $\mathcal{M}_q(P) = \{r \in \mathcal{M}(P) \ : \ |r| \le q\}$.

Denote by $B_{\tau p}(N, q)$ the union of all resource τp-bisimulations on $\mathcal{M}_q(P)$. Since the union of two resource τp-bisimulations is always a resource τp-bisimulation (Prop. 5.4) we obtain the largest resource τp-bisimulation on $\mathcal{M}_q(P)$.

Since $\mathcal{M}_q(P)$ is finite, we can use the weak transfer property to compute $B_{\tau p}(N, q)$.

Definition 14. *(Underapproximation of largest resource τp-bisimulation)*

Input: *a saturated labelled Petri net with invisible transitions $N = (P, T, F, l)$, parameter $q \in Nat$.*

Output: *Relation $B_{\tau p}(N, q)$.*

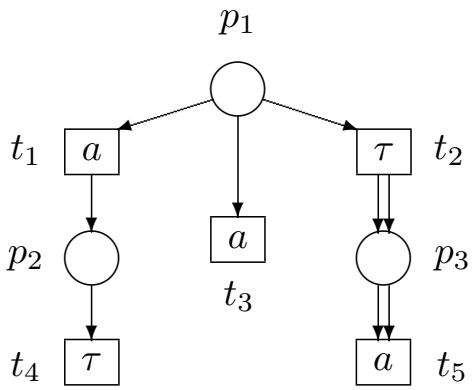
Step 1: *Let $C = \emptyset$ — an empty set of pairs (considered as a binary relation over $\mathcal{M}_q(P)$; it will be used as a set of discovered pairs of non-similar resources).*

Step 2: *Compute $B = (\mathcal{M}_q(P) \times \mathcal{M}_q(P)) \setminus C$. Since $\mathcal{M}_q(P)$ is finite the set of pairs B is also finite.*

Step 3: *Compute B_s — the ground basis of B.*

Step 4: *Check, whether B_s conforms to the weak τp-transfer property: it is sufficient to test all non-reflexive elements of B_s (denote a set of all non-reflexive elements of B_s by B_s^{nr}).*

- *If all pairs conforms to the weak τp-transfer property then stop and return B — the bisimulation.*
- *Otherwise there are $(r, s) \in B_s^{nr}$ and $t \in T$ with ${}^\bullet t \cap r \neq \emptyset$, s.t. the firing $M_1 \xrightarrow{t} M_1'$ with $M_1 = {}^\bullet t \cup r$ can not be imitated by a parallel step U with the same label and with precondition $M_2 = {}^\bullet t - r + s$ s.t. $M_2 \xrightarrow{U} M_2'$ with $(M_1', M_2') \in B^{AT}$. Add (r, s) and (s, r) to C and go back to Step 2.*



$$q = 1: \quad p_2 \approx_{\tau p} \emptyset$$

$$q = 2: \quad p_1 \approx_{\tau p} 2p_3$$

Fig. 6. An example of approximation: resource τp-bisimulation of a saturated Petri net with invisible transitions

Рис. 6. Пример аппроксимации: ресурс τp-бисимуляции насыщенной сети Петри с невидимыми переходами

(termination) For any marking the set of active parallel steps is finite. Also note that the set $\mathcal{M}_q(P) \times \mathcal{M}_q(P)$ is finite. Hence the algorithm always stops.

(correctness) Note that the algorithm stops only if B_s conforms to the weak τp-transfer property. Hence the result is always a resource τp-bisimulation.

(largest equivalence) Assume that not all pairs from the largest resource τp-bisimulation on $\mathcal{M}_q(P)$ are found. Hence each of the lost pairs was removed from the consideration (added to C) at some iteration of algorithm. Consider the first of these iterations. The pair is removed because it doesn't satisfy the weak τp-transfer property w.r.t. the current configuration of B_s. On the other hand, we know that it satisfies the weak τp-transfer property w.r.t. $B_{\tau p}(N, q)$. Since current iteration is first when we remove the "wrong" pair, it is clear that $B_{\tau p}(N, q) \subseteq (B_s)^{AT}$. Hence the pair of resources should satisfy the weak τp-transfer property w.r.t. $(B_s)^{AT}$ – a contradiction.

Denote by $\mathcal{R} = |\mathcal{M}_q(P)|$ the size of the set of considered resources.

At the Step 2 we search through the set of all parallel steps with at most one visible label, that can fire at marking M_2. Each invisible transition can participate in the parallel step at most $|M_2|$ times, since it uses at least one input token.[2] There is also at most one visible transition. Hence we have to check at most $|T||M_2|^{|T|}$ multisets of transitions.

The size of marking $M_2 = {}^\bullet t - r + s$ can be evaluated as $O(|s|) = O(q)$.

Using our previous estimations of complexity for ground basis calculation (polynomial w.r.t. $\mathcal{R}$) and the complexity of other steps of algorithm (polynomial w.r.t. the size of the net), we obtain the overall complexity of

$$O(\max\{|P|\ \mathcal{R}^9, |T|^2 q^{|T|} |P|\ \mathcal{R}^7\}).$$

Here the first and the second components of max are estimations for Step 3 and Step 4 respectively. So in the case of nets with invisible transitions the complexity of the algorithm increased significantly (the linear dependence on $|T|$ was replaced by an exponential one). Such a jump is explained by the transition from sets of transitions to multisets.

[2] Without loss of generality we can assume that a net contains no invisible transitions with empty preconditions. In any reachable marking an unobservable sequence of such generating transitions can increase the marking of any of their post-place to a value, exceeding any given natural number. Therefore the places that participate in the postconditions of such generating transitions actually do not affect the observable behavior of the net (and hence the bisimulations), and can be removed along with the corresponding generating transitions.

Consider an example of calculations (Fig. 6). Two subsequent steps are presented: $q = 1$ and $q = 2$. With $q = 1$ we found that resource p_2 is τp-similar to an empty resource (i.e. the place p_2 is redundant). Increasing the parameter ($q = 2$), we obtained one more pair of similar resources $p_1 \approx_{\tau p} 2p_3$.

Proposition 6. *Let $N = (P, T, F, l)$ be a saturated labelled Petri net with invisible transitions. Then:*

1. $\forall q \in Nat \quad (B_{\tau p}(N, q))^{AT} \subseteq (B_{\tau p}(N, q+1))^{AT}$;
2. $\exists q_f \in Nat : \quad \forall k \in Nat \quad B_{\tau p}(N, q_f + k) = B_{\tau p}(N)$.

Proof. (1) By construction of $B_{\tau p}(N, q)$ for any q the relation $(B_{\tau p}(N, q))^{AT}$ is a largest resource bisimulation s.t. the size of its generating elements (of ground basis) is not greater than q. The union of two resource bisimulations is also a resource bisimulation, hence $B' = (B_{\tau p}(N, q) \cup B_{\tau p}(N, q+1))^{AT}$ is a resource bisimulation. From the definition of ground basis the generating elements of B' have the size not greater than $q + 1$, therefore $B' = (B_{\tau p}(N, q+1))^{AT}$.

(2) Since any resource bisimulation is an AT-closed equivalence and therefore it has a finite ground basis (Th. 1). The value of q_f is the size of the largest element of the $B_{\tau p}(N)$ ground basis. □

So at some point q_f the sequence $\{B_{\tau p}(N, q)\}_q$ stabilizes. The problem of q_f computability is still open. The hypothesis is that τp-bisimulation of resources is undecidable and hence q_f is uncomputable.

5. On the approximation of τ-similarity in general nets

If a net is not saturable (see definition in Section 4.2), then the above procedure cannot be applied. However, some straightforward approximations still can be computed.

Consider a parameterized version of the weak τ-transfer property (Def. 6):

Definition 15. *Let $m, n \in Nat \cup \{\infty\}$. We say that a relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ conforms to the* (m, n)-weak τ-transfer property *if for all $(r, s) \in B^{AT}$, $t \in T$ s.t. $^\bullet t \cap r \neq \emptyset$ and $max\{|r|, |s|\} \leq m$, there exists an imitating sequence of transitions $\sigma \in T^*$ s.t. $l(t) =_\tau l(\sigma), |\sigma| \leq n$ and, denoting $M_1 = {}^\bullet t \cup r$ and $M_2 = {}^\bullet t - r + s$, we have $M_1 \xrightarrow{t} M_1'$ and $M_2 \xrightarrow{\sigma} M_2'$ with $(M_1', M_2') \in B^{AT}$.*

The first difference is that we check not only elements of B (the base elements of B^{AT}), but all elements of B^{AT} with at most m tokens. The second key property is that we simulate the transition firing not by an arbitrary sequence, but by a sequence with at most n transitions.

Definition 16. *A relation $B \subseteq \mathcal{M}(P) \times \mathcal{M}(P)$ is called an* (m, n)-equivalence *if both B and B^{-1} conform to the (m, n)-weak τ-transfer property.*

Definition 17. *Let N be a net with invisible transitions. Denote by $B_\tau^{(m,n)}(N)$ its largest (m, n)-equivalence.*

Proposition 7.

1. $B_\tau^{(0,0)}(N) = \mathcal{M}(P) \times \mathcal{M}(P)$.
2. $B_\tau^{(\infty,\infty)}(N) = B_\tau(N)$.

Proof. (1) From the definition of (m, n)-weak τ-transfer property.

(2) Note that in this case $(B_\tau^{(\infty,\infty)}(N))^{AT}$ conforms to the τ-transfer property, hence it is a marking τ-bisimulation. Moreover, it is the largest bisimulation since any union of marking bisimulations is a marking bisimulation. □

Obviously, the limit of sequence $\{B_\tau^{(m,n)}(N)\}_{m,n}$ for $m, n \longrightarrow \infty$ is $B_\tau(N)$. Consider two examples of such a sequence:

Example 1. *For the net depicted in Fig. 2 we have:*

$$B_\tau^{(1,1)}(N) = Id(P)$$

$$B_\tau^{(1,2)}(N) = Id(P) \cup \{(p_1, p_2), (p_2, p_1)\}$$

$$B_\tau^{(2,2)}(N) = Id(P) \cup \{(p_1, p_2), (p_2, p_1)\} \cup \{(p_i, p_j + p_k), (p_j + p_k, p_i) \mid i, j, k \in \{1, 2\})\}$$

$$\dots$$

$$B_\tau^{(m,n)}(N) = B_\tau^{(2,2)}(N)$$

$$\dots$$

$$B_\tau^{(\infty,\infty)}(N) = B_\tau^{(2,2)}(N)$$

Indeed, only the sequences of length 2 can find the similarity between p_1 and p_2. Hence (p_1, p_2) is added only on the second step. On the third step we find out that any non-empty multiset of places is equal to any other non-empty multiset of places — this can be defined by pairs $(p_i, p_j + p_k)$ and $(p_j + p_k, p_i)$ (all other elements can be obtained from these pairs and reflexive pairs with the help of an AT-closure). At the third step the sequence of sets stabilizes.

So as a result we have a non-contracting sequence:

$$(B_\tau^{(1,1)})^{AT} \subset (B_\tau^{(1,2)})^{AT} \subset (B_\tau^{(2,2)})^{AT} = \dots = (B_\tau^{(\infty,\infty)})^{AT}.$$

Example 2. *Consider the net depicted in Fig. 3. Here we have*

$$B_\tau^{(1,2)}(N) = Id(P) \cup \{(p_1, p_2), (p_2, p_1), (p_3, p_4), (p_4, p_3), (p_5, \emptyset), (\emptyset, p_5)\};$$

$$B_\tau^{(2,2)}(N) = Id(P) \cup \{(p_1 + p_4, p_2 + p_3), (p_2 + p_3, p_1 + p_4), (p_5, \emptyset), (\emptyset, p_5)\}.$$

Only at the second step the $(2, 2)$-weak τ-transfer property allowed us to discover the actual non-bisimilarity of resources p_1 and p_2.

The set of pairs is contracting in this particular case:

$$(B_\tau^{(1,2)})^{AT} \supset (B_\tau^{(2,2)})^{AT}.$$

Example 3. *Now consider a net, having two subnets – Fig. 2 and Fig. 3. Obviously, in this case*

$$(B_\tau^{(1,2)})^{AT} \begin{array}{c} \not\subseteq \\ \not\supseteq \end{array} (B_\tau^{(2,2)})^{AT}.$$

So, in general the sequence $\{B_\tau^{(m,n)}(N)\}_{m,n} \xrightarrow{m,n\longrightarrow\infty} B_\tau(N)$ is not monotonous even locally. Also note that $B_\tau^{(m,n)}(N)$ can be a subset of $B_\tau(N)$ (Example 1), a superset of $B_\tau(N)$ ($B_\tau^{(1,2)}(N)$ in Example 2) and incomparable to $B_\tau(N)$ (Example 3).

There are two open questions on the structure of $\{B_\tau^{(m,n)}(N)\}_{m,n}$ sequence:

1. Does it always stabilizes at some (m, n)?
2. If not, does it always become monotonous at some point (w.r.t. $m + n$)?

The hypothesis is that the answers are: (1) — negative, (2) — positive. The rationale for this is that $B_\tau^{(m,n)}$ is not always a bisimulation (in contrast to $B_{\tau p}(N, q)$ from the previous section) and hence the infinite "tail" of $\{B_\tau^{(m,n)}(N)\}_{m,n}$ can consist of an infinite sequence of contracting $B_\tau(N)$ overapproximations.

However, as it was shown in the previous examples, the (m, n)-equivalences can still be used in practice as non-trivial approximations of $B_\tau(N)$. The (m, n)-weak τ-transfer property can be effectively checked for any finitely-based candidate B (for example, defined by a ground base) and finite m and n.

Definition 18. *(Computation of an (m, n)-equivalence)*

Input: *a labelled Petri net with invisible transitions $N = (P, T, F, l)$, parameters $m, n \in Nat$.*

Output: *Relation $B_\tau^{(m,n)}(N)$.*

Step 1: *Compute a tree Tr of all possible ground bases (except the trivial reflexive basis $Id(P)$) having the size of their elements not greater than m. In this tree a basis B_s is a parent node for a basis B'_s iff $(B'_s)^{AT} \subset (B_s)^{AT}$.*

Step 2: *Using breadth-first search, take the next node B_s from Tr and check, whether B_s conforms to the (m, n)-weak τ-transfer property.*

- *If all pairs conforms to the (m, n)-weak τ-transfer property then stop and return B_s.*
- *Otherwise there are $(r, s) \in (B_s)^{AT}$ with $max\{|r|, |s|\} \leq m$ and $t \in T$ with ${}^\bullet t \cap r \neq \emptyset$, such that the firing $M_1 \xrightarrow{t} M_1{}'$ with $M_1 = {}^\bullet t \cup r$ can not be imitated by a sequence $\sigma \in T^*$ of (at most) n transitions with label $l(t)$ and precondition $M_2 = {}^\bullet t - r + s$ such that $M_2 \xrightarrow{\sigma} M_2{}'$ with $(M_1{}', M_2{}') \in B^{AT}$. In this case go back to the Step 2.*

Step 3: *Return $Id(P)$.*

(termination) The resource size is bounded by m, the length of firing sequences is bounded by n, the (m, n)-weak τ-transfer property can be checked in a finite number of steps. The tree Tr is also finite. Hence the algorithm always stops.

(correctness) The construction of the tree Tr implies that the largest (m, n)-equivalence is always the closest to the root (note that it contains all other (m, n)-equivalences). Hence the algorithm (breadth-first search) finds it first.

Note that this "algorithm" is simple, but highly ineffective. There are four non-polynomial procedures: Tr computation, Tr search, the resource pair combination and the transition sequence search.

6. Conclusion

The proposed methods for finding pairs of similar resources are of particular interest for certain applications, such as model reduction (shrinking the net without affecting its behavior) and adaptive process management (resource relocation in the aftermath of some acute events). In addition, the use of resource bisimulation allows one to reduce a Petri net with conservation of its behavior. This reduction is important when analyzing properties of the Petri net, since the computational complexity of the majority of algorithms used in analysis depends exponentially on the size of the net.

Important open questions concern decidability and complexity of related algorithmic problems. For example, we have already shown that all types of resource similarity (ordinary, τ-, τp-) are undecidable. On the other hand, the problem of $B(N)$ (and $B_\tau(N)$, and $B_{\tau p}(N)$) computability is still open. We have introduced only the underapproximations.

Acknowledgment. I would like to express my sincere gratitude to my mentor Irina Lomazova for her support and encouragement.

References

[1] R. Milner, "A Calculus of Communicating Systems", in, ser. Lecture Notes in Computer Science, vol. 92, Springer Berlin Heidelberg, 1980.

[2] D. Park, "Concurrency and automata on infinite sequences", in *Theoretical Computer Science*, ser. Lecture Notes in Computer Science, P. Deussen, Ed., vol. 104, Springer Berlin Heidelberg, 1981, pp. 167–183, ISBN: 978-3-540-10576-3. [Online]. Available: http://dx.doi.org/10.1007/BFb0017309.

[3] P. Jančar, "Decidability questions for bisimilarity of Petri nets and some related problems", in *STACS 94*, ser. Lecture Notes in Computer Science, P. Enjalbert, E. W. Mayr, and K. W. Wagner, Eds., vol. 775, Springer Berlin Heidelberg, 1994, pp. 581–592.

[4] C. Autant and P. Schnoebelen, "Place bisimulations in Petri nets", in *Application and Theory of Petri Nets 1992*, ser. Lecture Notes in Computer Science, K. Jensen, Ed., vol. 616, Springer Berlin Heidelberg, 1992, pp. 45–61, ISBN: 978-3-540-55676-3. [Online]. Available: http://dx.doi.org/10.1007/3-540-55676-1_3.

[5] C. Autant, W. Pfister, and P. Schnoebelen, "Place bisimulations for the reduction of labeled Petri nets with silent moves", in *Proc. 6th Int. Conf. on Computing and Information, Peterborough, Canada*, 1994.

[6] P. Schnoebelen and N. Sidorova, "Bisimulation and the reduction of Petri nets", in *Application and Theory of Petri Nets*, ser. Lecture Notes in Computer Science, vol. 1825, Springer Berlin Heidelberg, 2000, pp. 409–423.

[7] V. A. Bashkin and I. A. Lomazova, "Reduction of Coloured Petri nets based on resource bisimulation", *Joint Bulletin of NCC & IIS, Comp. Science*, vol. 13, pp. 12–17, 2000.

[8] V. A. Bashkin and I. A. Lomazova, "Petri nets and resource bisimulation", *Fundamenta Informaticae*, vol. 55, no. 2, pp. 101–114, 2003. [Online]. Available: http : / / iospress . metapress . com / content / NGX4LWFX81475D07.

[9] V. A. Bashkin and I. A. Lomazova, "Resource Similarities in Petri Net Models of Distributed Systems", in *Parallel Computing Technologies*, ser. Lecture Notes in Computer Science, V. E. Malyshkin, Ed., vol. 2763, Springer Berlin Heidelberg, 2003, pp. 35–48, ISBN: 978-3-540-40673-0. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-45145-7_4.

[10] I. A. Lomazova, "Resource Equivalences in Petri Nets", in *Application and Theory of Petri Nets and Concurrency*, ser. Lecture Notes in Computer Science, B. E. van der Aalst W., Ed., vol. 10258, Springer Berlin Heidelberg, 2017, pp. 19–34.

[11] V. A. Bashkin, "On the Resource Equivalences in Petri Nets with Invisible Transitions", in *PNSE and Petri Nets 2017*, ser. CEUR-WS, vol. 1846, 2017, pp. 51–68.

[12] L. Redei, *The theory of finitely generated commutative semigroups*. Oxford University Press, 1965.

[13] Y. Hirshfeld, "Congruences in commutative semigroups", Edinburgh, University of Edinburgh, Department of Computer Science, Research Report ECS-LFCS-94-291, 1994.