ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук

Образовательная программа «Прикладная математика и информатика»

Отчет о программном проекте

на тему: Программа для исправления сетей Петри с использованием гамаков

(промежуточный, этап 1)

Выполнил:

Студент группы БПМИ ___ _____ _____

Подпись И.О.Фамилия

15.02.2022

Дата

Принял:

Руководитель проекта Алексей Александрович Мицюк

Имя, Отчество, Фамилия

заведующий кафедрой, доцент, канд. комп. наук

Должность, ученое звание

Базовая кафедра компании JetBrains ФКН НИУ ВШЭ

Место работы (Компания или подразделение НИУ ВШЭ)

Дата проверки 17.02 2022 _____ _____ _____

Оценка (по 10-ти бальной шкале) Подпись

Москва 2022

Содержание

Основные термины, определения и сокращения

Модель процесса (process model) — формальное математическое описание некоторого, зачастую естественного, процесса. Будем считать, что для рассматриваемого процесса выделено мн-во *действий*(activities) и *названий действий* (activity labels), входящих в него. Мн-во названий действий в контексте рассматриваемого процесса далее обозначается как $\mathcal{A}$.

Журнал или **лог событий** (event log) [в простом смысле]: Пусть $\mathcal{A}$ — мн-во названий действий некоторого процесса. Будем называть *следом*(trace) произвольную последовательность действий из $\mathcal{A}$, а *журналом (логом) событий* — мультимножество следов.

Сеть Петри (Petri net) — тройка $N = (P, T, F)$, где P(places) — конечное мн-во *позиций*, T(transitions) — конечное мн-во *переходов* такое, что $P \cap T = \varnothing$, и

F(flow relation)$\subseteq (P \times T) \cup (T \times P)$ — мн-во ориентированных рёбер между позициями и переходами.

Маркированная сеть Петри (marked Petri net) — пара (N, M), где $N = (P, T, F)$ — сеть Петри, а M(marking) $\in \mathbb{B}(P)$ — мультимножество на P, называемое *маркировкой* и зачастую интерпретируемое как мн-во *токенов* в сети.

Размеченная сеть Петри (labeled Petri net) — набор $N = (P, T, F, A, l)$, где (P, T, F) — сеть Петри, $A \subseteq \mathcal{A}$ — подмножество названий действий процесса, и $l : T \mapsto A$ — функция разметки, сопоставляющая переходам названия действия.

Пусть $N = (P, T, F)$ – сеть Петри. Будем рассматривать мн-во $P \cup T$ как мн-во вершин сети, как в ориентированном графе с рёбрами F (это подразумевается далее при рассмотрении сети Петри как орграфа). Для вершины v обозначим мн-во соседних с ней по входящим рёбрам вершин как $\bullet v$, а мн-во соседних по исходящим как $v\bullet$.

Сеть рабочего процесса (workflow net): Пусть $N = (P, T, F, A, l)$ – размеченная сеть Петри и $\bar{t} \notin P \cup T$. Тогда N называется *сетью рабочего процесса*, если

a) $\exists i \in P$ такая, что $\bullet i = \varnothing$; позиция i называется *начальной* или *истоком*

b) $\exists o \in P$ такая, что $o\bullet = \varnothing$; позиция o называется *конечной* или *стоком*

c) Сеть $\bar{N} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\}, A \cup \{\tau\}, l \cup \{(\bar{t}, \tau)\}$ сильно связна как орграф

Граф потока управления (control flow graph) — набор $G = (V, E, v_s, v_t)$, где (V, E) – ориентированный граф с мн-вом вершин V и мн-вом рёбер E, где в каждую вершину есть путь из $v_s \in V$, и из каждой вершины есть путь в $v_t \in V$.

Гамак (hammock): пусть (V, E, v_s, v_t) – граф потока управления. Тогда его подграф $H = (V', E', s, t)$ называется *гамаком* с истоком $s \in V'$ и стоком $t \in V'$, если

$\forall e = (u, v) \in E,\ u \notin V',\ v \in V' \Rightarrow v = s$

$\forall e = (u, v) \in E,\ u \in V',\ v \notin V' \Rightarrow u = t$

Неформально говоря, все входящие извне в гамак рёбра входят в исток, а все исходящие из него рёбра выходят из стока.

Заметим, что сеть рабочего процесса $N = (P, T, F, A, l)$ с начальной позицией i и конечной позицией o, если рассматривать её как орграф, является графом потока управления $G = (P \cup T, F, i, o)$. Таким образом, приведённое определение гамака применимо и в отношении сетей рабочего процесса (какими и будут являться в большинстве случаев рассматриваемые сети Петри).

Введение

В последние годы наблюдается стремительное развитие науки о данных (data science). Одним из наиболее распространённых способов её приложения является изучение и анализ бизнес-процессов. Совмещая в себе использование данных от data science и построение классических моделей от business process management (BPM), относительно недавно, в качестве самостоятельной области зародилась теория process mining (в некоторых источниках название адаптируется как *процессная аналитика*). Process mining предлагает ряд методов и подходов для анализа и улучшения бизнес-процессов путём построения их модели и её сравнения с реальным журналом событий.

Само понятие process mining было введено немецким учёным в области компьютерных наук – проф. Вилом ван дер Аалстом. Основой данной теории выступает его книга [1], используемая в качестве главного источника в ходе работы над данным проектом (в том числе, большинство приведённых определений были взяты из неё).

Основными направлениями process mining являются:

- *Обнаружение процесса* (process discovery) — построение модели процесса на основании его журнала событий
- *Проверка соответствия* (conformance checking) — сравнение существующей модели и журнала событий с целью выявления несоответствий между ними
- *Улучшение* (enhancement) — изменение существующей модели на основании журнала событий, разделяемое на *исправление* (repair) – когда изменения нацелены на её большее соответствие реальному процессу, и *расширение* (extension) – когда изменения нацелены на отражение новой информации моделью, например, о времени, затрачиваемым на действия

Модель процесса может быть представлена множеством различных способов. Одним из самых распространённых и, наверное, естественных из них является представление в виде сети Петри.

На рис. 1 представлен пример модели процесса в виде размеченной сети Петри с единственным токеном в начальной позиции, квадраты соответствуют переходам, круги – позициям. Кратко и не совсем формально опишем симуляцию процесса в ней:

Опр. (Правило активации перехода)**.** Пусть (N, M) – маркированная сеть Петри, $N = (P, T, F)$. Будем называть переход $t \in T$ *включённым*, если $\bullet t \subseteq M$.

Результатом *активации* включённого перехода t будем считать маркированную сеть Петри (N, M'), где $M' = (M \setminus (\bullet t)) \cup (t\bullet)$

Токены $\bullet t$ будем называть *поглощёнными* (consumed), а $t\bullet$ – *произведёнными* (produced).

Последовательность переходов $\sigma = (t_1, \ldots, t_k), t_i \in T$ будем называть *последовательностью активаций*,

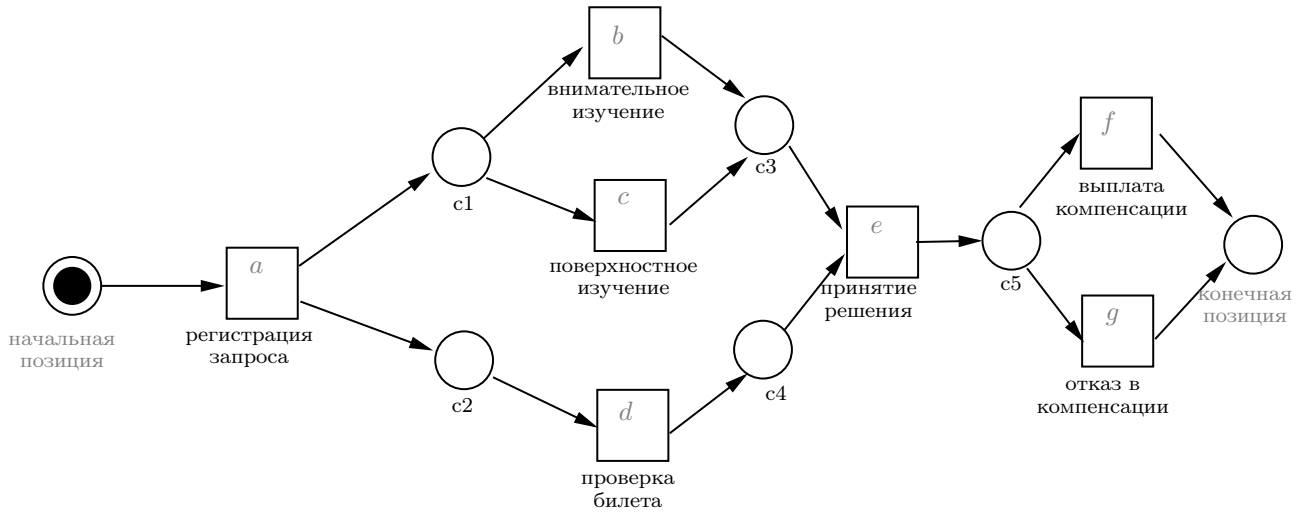


Рис. 1: *Пример модели процесса оформления компенсации за авиабилет в виде сети Петри.*

если $\forall i : t_i$ включён в (N, M_{i-1}), где (N, M_{i-1}) — результат последовательной активации переходов $t_1, \ldots, t_{i-1}$ в (N, M), $M_0 = M$.

Например, в приведённой на рис. 1 сети включённым переходом является только (`регистрация запроса`). После его активации мн-во токенов станет равно $M' = \{c_1, c_2\}$, а включёнными переходами станут (`внимательное изучение`), (`поверхностное изучение`) и (`проверка билета`).

Опр. (Воспроизводимость и соответствие следа)**.** Будем называть след $\sigma = \langle a_1, \ldots, a_k \rangle$ из журнала событий *воспроизводимым* для размеченной маркированной сети Петри $(N, M), N = (P, T, F, A, l)$, если $\exists(t_1, \ldots, t_k)$ – посл-ть активаций в сети такая, что $\forall i : l(t_i) = a_i \in A$.

Будем считать, что след *соответствует* модели, если дополнительно полученная в результате последовательной активации переходов $t_1, \ldots, t_k$ маркировка M' равна M_f для некоторой заданной финальной маркировки M_f.

В случае отсутствия неоднозначности будем использовать одинаковые обозначения для действий и переходов.

Таким образом, *соответствующими* модели на рис. 1 следами при финальной маркировке {`конечная позиция`} являются, например, $\langle a, b, d, e, f \rangle$ и $\langle a, d, c, e, g \rangle$. В то время как не соответствуют ей следы $\langle a, b, d, e \rangle$ (полученная маркировка не будет совпадать с финальной) и $\langle a, b, c, d, e, f \rangle$ (не является воспроизводимым).

С помощью методов проверки соответствия можно находить «проблемные» переходы в построенной модели по некоторым критериям. Возникает интуитивное предположение о том, что, используя эту информацию, можно выделять относительно независимые и изолированные структурные единицы сети (в некотором

смысле, подсети), а затем перестраивать их с помощью методов обнаружения процесса на основании соответствующих частей записей в журнале событий. Подобный подход встречается в статьях (например, в [2], откуда было взято определение гамака), описывающих структурирование и рефакторинг программного кода путём выделения гамаков при его интерпретации в виде графа потока управления. Предполагается, что можно аналогично выделять гамаки и в случае исправления моделей, представленных в виде сети Петри.

Целью проекта является разработка алгоритма исправления моделей, представленных в виде сети Петри, который основывается на выделении гамаков в ней, и его реализация в качестве части open-source python-библиотеки PM4PY [3]

Задачами проекта, ведущими к этой цели, являются:

- Изучение основной теории process mining
- Анализ существующих методов проверки соответствия и выбор наиболее подходящего для предлагаемого алгоритма. Интерпретация результата его применения для последующего исправления модели
- Создание/нахождение алгоритма выделения гамаков в сети Петри
- Реализация прототипа алгоритма исправления модели
- Оценка качества работы алгоритма относительно ожидаемых результатов и других алгоритмов на основании журналов событий, как сгенерированных по самостоятельно составленным моделям, так и размещённых в свободном доступе
- Возможные изменения в подходе или используемых методах в зависимости от полученных промежуточных результатов
- Структурирование и приведение к финальному виду теоретических выкладок, алгоритма, его реализации; тестирование реализации
- Работа над включением реализации алгоритма в библиотеку PM4PY [3]

Требования к программе

Функциональные

Результатом проекта должна стать реализация описанного во введении алгоритма в качестве отдельного модуля python-библиотеки PM4PY [3], исходный код которой размещён в GitHub-репозитории [1]. Основной должна быть функция, вызываемая от сети Петри, задающей модель процесса, и журнала событий, представленных во внутреннем формате PM4PY, с возможными параметрами. Результатом работы функции должна быть новая сеть Петри, исправленная на основании предоставленного журнала.

Реализация должна следовать общему стилю и уже существующей логике библиотеки. В коде должны использоваться имеющиеся внутренние функции работы с сетями/журналами событий вместо самописных везде, где это возможно. Возможно внесение изменений в функции библиотеки с целью необходимого расширения их функциональности.

Функция должна проверять все необходимые для применения алгоритма инварианты и сообщать об их невыполнении или возникших из-за неправильных аргументов ошибках. Она должна корректно работать при передаче любых аргументах, удовлетворяющим требованиям алгоритма. Для проверки этого должны быть реализованы тесты, проверяющие все части реализации как на корректность логики, например, при работе с крайними случаями входа, так и на корректность работы с неправильными данными.

Реальное время работы алгоритма на входных данных разумного размера не должно превышать времени работы сравнимых операций работы с моделями и журналами событий, реализованных в библиотеке.

Нефункциональные

Реализованный алгоритм должен иметь математически корректное строгое теоретическое обоснование. Должны быть приведены асимптотическая оценка его вычислительной сложности и требуемой дополнительной памяти. Эти оценки должны быть заметно лучше, чем у алгоритмов обнаружения процесса, которые могут быть использованы для полного построения новой модели. Желательно приведение примеров входных данных, на которых достигается худшая оценка.

[1] GitHub репозиторий библиотеки PM4PY – `https://github.com/pm4py`

Обзор и сравнительный анализ

Обзор результатов работы

Изученные материалы

Была изучена основная теория по process mining, представленная в книге [1] – вводимые определения, α-алгоритм обнаружения процесса, представленные алгоритмы проверки соответствия: воспроизведение следов с помощью токенов в сети Петри (token-based replay); метод сравнения моделей, журналов событий как объектов одного типа после построения определённых таблиц по ним (footprints); нахождение наиболее похожих на заданный воспроизводимых следов в модели (построение alignments). Была опробована библиотека PM4PY в работе с журналами событий и сетями Петри.

Выбор используемого метода проверки соответствия

Было решено для реализуемого алгоритма использовать именно метод построения alignments, детально описанный в [1, Раздел 8.3]. Кратко и неформально опишем его:

Суть метода заключается в поиске для заданного следа $\sigma = \langle a_1, \ldots, a_k \rangle$ «оптимального» относительно него и модели следа, воспроизводимого рассматриваемой моделью. Пусть след $\sigma = \langle a, d, b, e, h \rangle$ является воспроизводимым в модели. Тогда, что вполне естественно, единственным оптимальным соответствием (alignment) является:

след	a	d	b	e	h
путь в модели	a	d	b	e	h

, где верхняя строка соответствует следу σ, а нижняя – оптимальному воспроизводимому следу (на это можно смотреть, как на посл-ть активаций переходов в сети, задающей модель).

Для случая, когда след не является воспроизводимым, вводится символ "$\gg$" , означающий пропуск. Например, в следующем alignment для того же следа (но для некоторой другой модели) в качестве оптимального был найден след $\langle a, b, d, e, h \rangle$:

след	a	$\gg$	d	b	e	h
путь в модели	a	b	d	$\gg$	e	h

1-ая пара (a, a) означает *синхронный* шаг (sync move) – активацию перехода в сети, соответствующего действию в следе; далее пара $(\gg, b)$ означает шаг *только в модели* (model move), т.е. активации перехода b в модели не соответствует действие в следе; (d, d) – снова синхронный шаг; $(b, \gg)$ означает шаг *только в логе* (следе) (log move), т.е. на соответствующее действие в следе в модели не было активаций переходов; далее вновь следуют синхронные шаги (e, e) и (h, h).

Для сравнения alignments по «оптимальности» вводится функция стоимости, которая зависит от длины alignment, кол-ва ходов только в логе/только в модели (причём веса у различных типов отклонений могут быть разными). Таким образом, оптимальных alignments может быть несколько.

По построенным alignments для каждого следа в журнале событий можно считать различные характеристики, измеряющие отклонение модели от журнала.

Этот метод был выбран, потому что он позволяет выделять последовательности переходов, которые выполнялись в соответствии со следом – подряд идущие синхронные шаги, а также «проблемные» переходы, например, те, которые встречаются в шагах только в модели/логе. Разбиение на «плохие» и «хорошие» отрезки в alignments интуитивно соответствует локализации не соответствующих журналу участков сети, и его предлагается использовать для выделения гамаков.

Также в PM4PY уже реализованы функции для нахождения оптимальных alignments, поэтому их можно легко использовать в реализации алгоритма.

Использование результатов проверки соответствия

Был предложен алгоритм использования найденных alignments.

Хочется выделять пары вершин в сети, которые будут соответствовать началу и концу «проблемного» участка, а затем искать гамаки таким образом, чтобы каждая такая пара содержалась в одном гамаке. Наивный подход следующий:

> Выделим в alignment все максимальные по длине последовательности подряд идущих синхронных шагов. Выбросим из рассмотрения те последовательности, которые состоят из одного шага, если только этот шаг не первый и не последний в порядке alignment (предполагаем, что первый и последний шаги всегда синхронные). Будем считать оставшиеся последовательности «хорошими» участками, их правые концы – началом, а левые – концом «плохих» участков в порядке alignment (пример выделения пар на рис. 2).

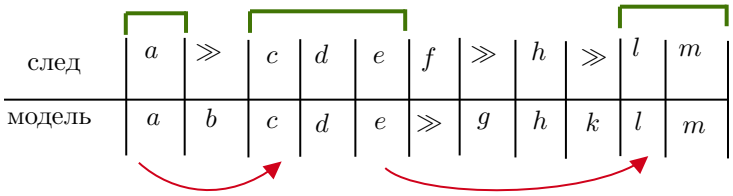


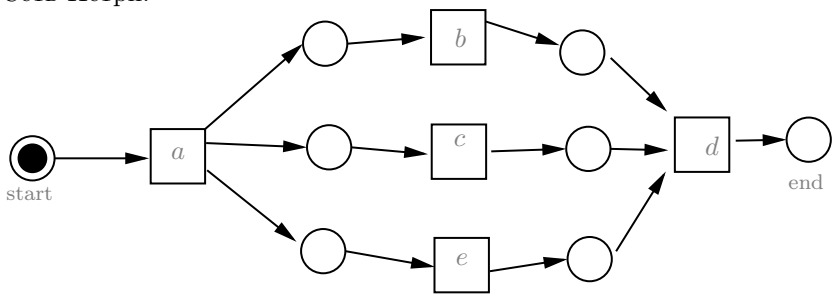
Рис. 2: *Пример наивного использования alignment для выделения пар. Зелёным отмечены «хорошие» участки, красные рёбра соответствуют парам выделенных вершин.*

> После выделения таких пар для alignment по каждому следу из журнала предлагается выбрать пары, которые встречались наиболее часто.

Однако данный подход может приводить к выделению большого числа пар, которые на самом деле не соответствуют «проблемным» участкам сети, даже при появлении малейшей параллельности. Что ожидаемо, ведь при выделении пар явно используется линейный порядок alignment без привязки к структуре сети, а при

возникающей параллельности действия из разных веток могут быть упорядочены в alignment произвольным образом. Рассмотрим пример:

Сеть Петри:



В следах $\sigma_1 = \langle a, b, e, d \rangle$ и $\sigma_2 = \langle a, e, b, d \rangle$ фактически одно и то же несоответствие с моделью – отсутствует действие c. Возможные оптимальные alignments:

след	a	b	≫	e	d
модель	a	b	c	e	d

след	a	≫	b	e	d
модель	a	c	b	e	d

след	a	e	≫	b	d
модель	a	e	c	b	d

след	a	≫	e	b	d
модель	a	c	e	b	d

И описанный алгоритм может выделить 4 различных пары вершин: $(b, e),\ (a, b),\ (e, b),\ (a, e)$. При большем размере сети кол-во возможных пар увеличится, и какие-то из них могут и не находиться в действительно «проблемных» участках сети, что и было замечено при применении алгоритма на примерах.

Для улучшения наивного подхода предлагается использовать воспроизведение следов с помощью токенов с целью использования структуры сети:

Для каждого токена t будем хранить мн-во предков $A(t)$ – последних переходов, при активации которых токен был поглощён (*пройденных* переходов), и $B(t)$ – мн-во последних пройденных переходов, которые соответствовали синхронным шагам в alignment. Изначально у всех токенов оба мн-ва пустые.

Будем последовательно идти по шагам alignment. Шаги только в логе будем просто пропускать (что далее будет пояснено). Пусть $t_1, \ldots, t_k$ – поглощённые токены при активируемом переходе T_a, а t_p – новый токен, которой будет произведён:

- шаг только в модели:

  Если T_a – *скрытый* переход (hidden transition):

  Положим $A(t_p) = \bigcup_{i=1}^{k} A(t_i),\ B(t_p) = \bigcup_{i=1}^{k} B(t_i)$

  Иначе:

  Положим $A(t_p) = \varnothing,\ B(t_p) = \left(\bigcup_{i=1}^{k} A(t_i)\right) \cup \left(\bigcup_{i=1}^{k} B(t_i)\right)$

- синхронный шаг:

  Выделим пары: $(T, T_a)\ \forall T \in \bigcup_{i=1}^{k} B(t_i)$

  Положим $A(t_p) = \{T\},\ B(t_p) = \varnothing$

В предложенном алгоритме шаги только в логе пропускаются по причине того, что нельзя однозначно определить, с каким текущим токеном/переходом связано это несоответствие. Например, если в ходе активации переходов в сети находятся несколько токенов в различных позициях, и следующим в alignment является шаг только в логе, то невозможно соотнести данный шаг с каким-то одним из этих токенов. Может показаться естественным соотнести несоответствие с предыдущим переходом в alignment, однако, как упоминалось в описании наивного подхода, порядок последовательности шагов не может гарантировать, что в модели предыдущий переход действительно должен предшествовать текущему действию из следа.

«Черновой» (т.к., возможно, подход к выделению пар будет изменён в ходе реализации алгоритма выделения гамаков и их замены) вариант реализации описанного подхода вместе с некоторой простой визуализацией приведён в python-ноутбуке [2].

Нахождение гамаков в сети Петри

В статье [2], где приводится используемое определение гамака, описан алгоритм его поиска лишь в контексте рассматриваемой задачи (связанной с улучшением структуры программного кода), и он не может быть использован в общем случае для произвольного графа потока управления.

Был предложен алгоритм нахождения наименьшего гамака, покрывающего заданное мн-во вершин в графе. Приведём его описание с набросками некоторых доказательств. Работа над алгоритмом всё ещё ведётся, поэтому обоснование является неполным (и возможно, что какие-то из утверждений являются неверными).

Пусть $G = (N, E, n_s, n_t)$ – некоторый граф потока управления.

Вопрос: Существует ли вообще наименьший гамак, содержащий выбранное мн-во вершин?

Утв. *Пересечение гамаков является гамаком (с некоторыми истоком и стоком)*

Тогда, т.к. всегда найдётся хотя бы один покрывающий выбранное мн-во гамак $H = (N, E, n_s, n_t)$, всегда найдётся такой наименьший по включению.

Введём ряд обозначений:

$N_c \subseteq N$ – выбранное мн-во вершин, которые должны накрываться гамаком

$P_t(V), V \subseteq N$ – мн-во всех вершинно-простых путей из вершин V в n_t

$P_s(V), V \subseteq N$ – мн-во всех вершинно-простых путей из вершин V в n_s в транспонированном графе G^T

Для некоторого мн-ва множеств X обозначим $I(X) = \bigcap_{x \in X} x$

[2] «Черновой» вариант выделения пар с визуализацией – `https://github.com/TrickmanOff/hammocks_repair/blob/main/BadPairsClean.ipynb`

Утв 1. *Пусть $V \subseteq N$ – некоторое подмн-во вершин, $t \in N$ – выбранная вершина, P – мн-во вершинно-простых путей из вершин V в вершину t. Тогда вершины $I(P)$ можно строго упорядочить так, чтобы они входили в каждый путь из P в определённом порядке.*

□

Заметим, что мн-во $I(P)$ не пусто, т.к. в него, конечно, входит n_t. Предположим, что утверждение неверно. Тогда найдутся две вершины $v_1 \neq v_2 \in I(P)$ такие, что $\exists\ p_1 \neq p_2 \in P : p_1 = (v_i, \dots, v_1, \dots, v_2, \dots, t), v_i \in V$, $p_2 = (v_j, \dots, v_2, \dots v_1, \dots, t), v_j \in V$.

Тогда можно выбрать некоторый префикс p_1 не более $(v_i, \dots, v_1)$ и суффикс p_2 не более $(v_1, \dots, t)$ такие, что, совместив их, получим вершинно-простой путь $p' = (v_i, \dots, t) \in P, v_2 \notin p'$, что противоречит тому, что $v_2 \in I(P)$.

■

Тогда и вершины из $I(P_s(N_c))$ и $I(P_t(N_c))$ можно аналогично упорядочить. Естественным «кандидатом» на исток гамака s является 1-ая вершина в таком порядке из $I(P_s(N_c))$, а «кандидатом» на сток t – из $I(P_t(N_C))$ (заметим, что они они могут ими и не являться).

Заметим, что в условиях утв. 1 $|I(P) \cap V| \leq 1$, потому что иначе, если в пересечении есть две вершины $v_1 <_o v_2$ (где $<_0$ – отношение порядка на вершинах из $I(P)$), то $v_1 \notin I(P)$, т.к. есть вершинно-простой путь из $v_2 \in V$ в t, не содержащий v_1.

Алгоритм поиска наименьшего гамака, содержащего N_c:

Положим сначала $V = N_c$ и будем далее расширять это мн-во, пока оно не станет гамаком.

Шаг алгоритма:

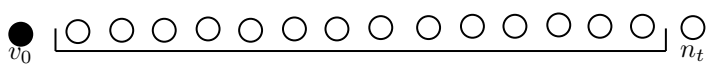
Пусть s', t' – найденные упомянутые кандидаты на исток и сток гамака, содержащего V. При добавлении новых вершин в V s' и t' могут измениться только на эти новые вершины (а могут и остаться прежними). Тогда все вершины $V \setminus \{s', t'\}$ уже включены в искомый наименьший гамак и не могут являться в нём ни истоком, ни стоком (потому что эти вершины уже не попадут в $I(P_*(V))$, а исток и сток точно там лежат). Включим всех соседей этих вершин (по входящим и исходящим рёбрам) в V, они точно попадут в гамак. Также включим всех соседей s' по исходящим рёбрам и соседей t' по входящим. Если мн-во добавленных вершин не пусто, то повторим шаг.

Иначе мн-во V является искомым гамаком с истоком $s = s'$, и стоком $t = t'$.

Остаётся лишь научиться искать s' и t'.

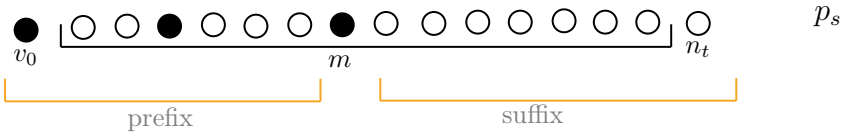
Опишем поиск t', поиск s' симметричен:

Рассмотрим *произвольный* путь из $P_t(V)$ – p_s. Введём мн-во *отмеченных* вершин, в которое изначально будет входить лишь первая вершина пути p_s – $v_0 \in V$.



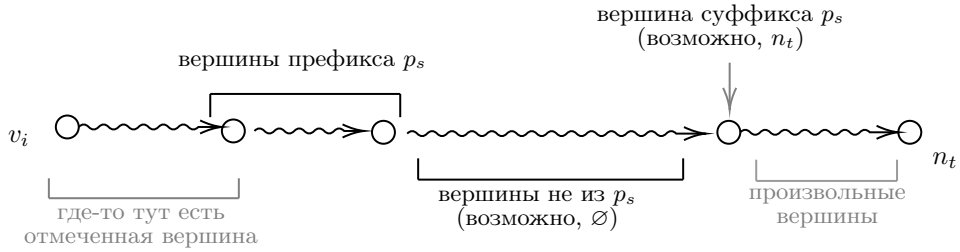
Запустим обходы из остальных вершин V. При входе в некоторую вершину p_s (а этот вход произойдёт для любого пути) будем *отмечать* эту вершину и не будем продолжать из неё обход.

Пусть m – последняя отмеченная вершина p_s в порядке обхода пути. Назовём *префиксом* пути p_s все вершины строго до неё, а *суффиксом* – строго после (оба могут быть пусты). Заметим, что никакая вершина префикса не может лежать в $I(P_t(V))$. Будем в ходе алгоритма поддерживать данный инвариант.



Если все пути из $P_t(V)$ проходят через вершину m, то она лежит в $I(P_t(V))$ и, согласно инварианту префикса, является искомой t'.

Если нет, то рассмотрим пути, не проходящие через m. Они имеют следующий вид:



Никакая вершина из p_s до достижимых таким образом вершин суффикса не может лежать в $I(P_t(V))$.

Запустим обход из вершин префикса по вершинам не из p_s, отмечая первые встреченные вершины суффикса и не продолжая из них обход (в таком случае позиция вершины m сдвигается вправо, и соответствующий префикс p_s увеличивается с сохранением инварианта). Будем продолжать обходы вершин префикса. Оставшаяся в конце вершина m и будет искомой.

Кажется, что для обхода из всех вершин префикса можно делать как бы один обход из них, сохраняя общее состояние посещённых вершин. Таким образом, поиск t', как и s', будет осуществляться за $O\big(|N| + |E|\big)$, а весь алгоритм поиска гамака – за $O\big(|N|(|N| + |E|)\big) = O\big(|N| \cdot |E|\big)$

Сравнительный анализ аналогов

В книге [1] не описаны алгоритмы улучшения моделей.

В статье [4] предложен алгоритм исправления моделей, представленных в виде сети Петри, и детально описывается его последовательное улучшение. В нём для нахождения проблемных переходов также используется построение alignments. Для шагов только в модели применяется достаточно простой метод исправления,

а для шагов только в логе из следов выделяются соответствующие участки, по которым строятся так называемые *подпроцессы*, встраиваемые в соответствующие места сети. Однако алгоритм опирается на порядок следования шагов в alignment, из-за чего, как предполагается, могут возникнуть некоторые неточности при исправлении сети. Возможно использование для ходов только в логе предлагаемого в статье варианта исправления, а для ходов только в модели – алгоритма, основанного на выделении гамаков.

В целом, существует лишь небольшое число методов исправления моделей. Среди них не было найдено тех, которые используют выделение гамаков в сети Петри, поэтому можно ожидать, что данный подход будет иметь положительные особенности относительно других.

В библиотеке PM4PY на данный момент не реализованы алгоритмы исправления сетей. Для фреймворка ProM [5] существуют пакеты, например, Uma – с реализацией одного из алгоритмов из статьи [4], и плагины с реализацией алгоритмов исправления моделей.

Календарный план

задача	**ожидаемая дата выполнения**
Приведение и математическое обоснование алгоритма выделения гамаков в сети Петри, задающей модель процесса	17.02.2022
Реализация прототипа алгоритма model repair	05.03.2022
Оценка качества работы алгоритма, принятие решения о его применимости/ необходимости улучшения	20.03.2022
Завершение исправлений и улучшений алгоритма, получение итоговых метрик качества результата его работы	05.04.2022
Приведение теоретических выкладок, кода к финальному виду, готовому для использования в целевой библиотеке. Написание документации.	20.04.2022
Включение реализации алгоритма в библиотеку PM4PY	20.05.2022

Список источников

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

[2] Erik H. D'Hollander Fubo Zhang. «Using Hammock Graphs to Structure Programs». в: *IEEE Transactions on Software Engineering* 30.4 (апр. 2004).

[3] Process Mining Group of the Fraunhofer Institute for Applied Information Technology. *PM4PY*. `https://pm4py.fit.fraunhofer.de/`. Посещён 02-02-2022.

[4] Wil M.P. van der Aalst Dirk Fahland. «Model Repair — Aligning Process Models to Reality». в: *Information Systems* 47 (янв. 2015).

[5] *ProM - Framework for process mining*. `https://www.promtools.org/`. Посещён 02-02-2022.