Факультет компьютерных наук

National Research University Higher School of Economics

программный проект

Программа для исправления сетей Петри с использованием гамаков

Program for Petri Net Repair using Hammocks

выполнил: Егоров Егор Александрович, студент БПМИ201-2

руководитель: Мицюк Алексей Александрович, канд. комп. наук, доцент, заведующий базовой кафедрой компании JetBrains ФКН НИУ ВШЭ

Предметная область

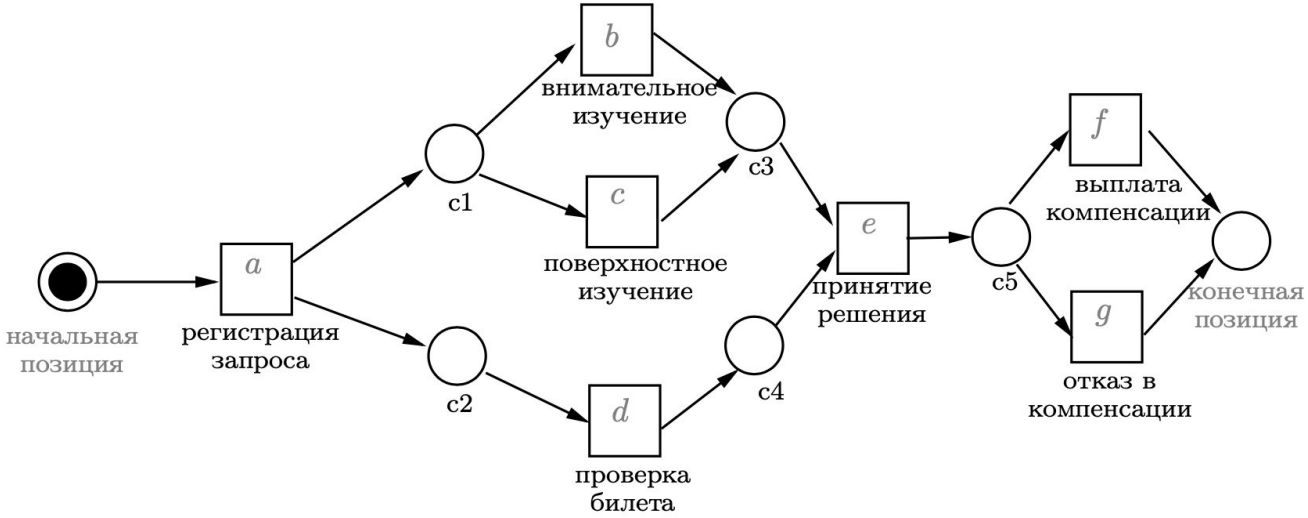
Process mining - область, находящаяся на стыке Data Science и анализа бизнес-процессов. Её объектом является анализ улучшения бизнес-процессов путём построения их моделей и сравнения этих моделей с журналом событий (логом).

Основными задачами process mining являются:

- *Обнаружение процесса* (process discovery)
- *Проверка соответствия* (conformance checking)
- *Улучшение модели* (model enhancement), разделяемое на *исправление* (repair) и *расширение* (extension)

Основой теории по process mining в рамках работы над проектом выступала книга проф. Вила ван дер Аалста: *Process Mining: Data Science in Action* [1]

Предметная область



Воспроизводимые посл-ти действий (*следы*):
<a, b, d, e, f>
<a, d, c, e, g>

Пример модели процесса оформления компенсации за авиабилет в виде сети Петри.

Цель

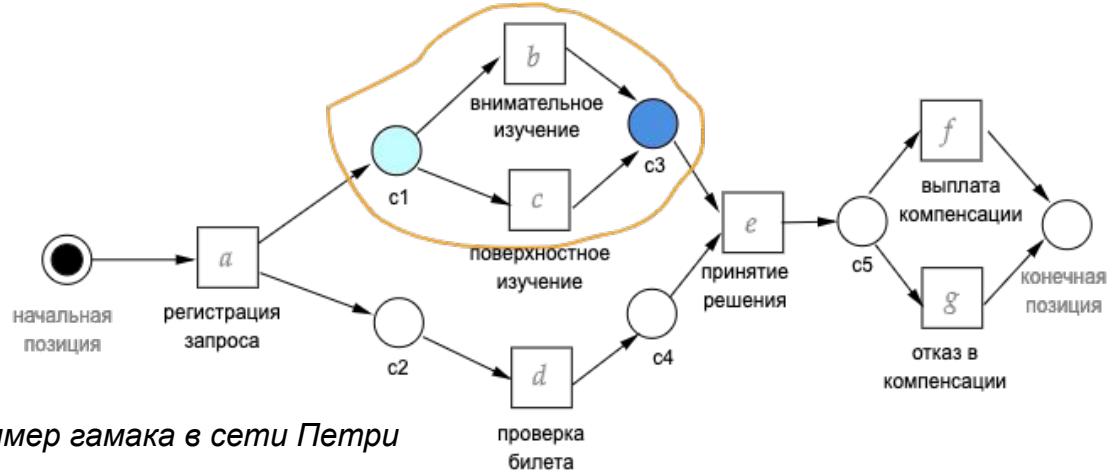
Цель: разработка алгоритма исправления моделей, представленных в виде сети Петри, который основывается на выделении *гамаков* в ней, и его реализация в качестве части open-source python-библиотеки PM4PY[1].

[1] *GitHub-репозиторий библиотеки PM4PY:* https://github.com/pm4py - [3]

Цель

Цель: 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[1].

В ориентированном графе *гамак* - такой подграф, что все входящие извне в него рёбра входят в некоторую одну вершину (*исток*), и все исходящие из него наружу рёбра выходят из некоторой одной вершины (*стока*).



пример гамака в сети Петри

Если в сети присутствуют какие-то локальные несоответствия логу, то интуитивно гамак, содержащий “проблемные” вершины, будет соответствовать некоторому подпроцессу, который можно легко вырезать из сети и заменить на исправленный каким-то простым способом.

[1] *GitHub-репозиторий библиотеки PM4PY:* https://github.com/pm4py - [3]

Задачи

- Изучение основной теории process mining
- Выбор метода *проверки соответствия* для выделения “проблемных” вершин
- Разработка алгоритма выделения гамаков в сети Петри
- Реализация прототипа алгоритма исправления модели
- Оценка качества работы алгоритма

Анализ существующих методов
Актуальность работы

В библиотеке PM4PY на момент работы над проектом не реализованы алгоритмы исправления сетей.

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. Ожидается, что использование гамаков позволит добиться “похожести” исправленной сети на исходную и будет показывать хороший результат при небольших локальных несоответствиях сети.

К тому же, выделение гамаков используется в некотором смысле похожим образом для выделения изолированных частей программ для их дальнейшей оптимизации и реструктуризации[2].

Функциональность существующих методов исправления представляет собой (что ожидаемо) построение исправленной сети по исходной сети и логу с некоторой параметризацией.

Функциональные требования

- результатом проекта должна стать реализация описанного во введении алгоритма в качестве части Python-библиотеки PM4PY
- реализация должна следовать общему стилю и уже существующей логике библиотеки
- основной должна быть функция вида repair(Petri net, Event log, Parameters) -> Petri net
- время работы алгоритма на входных данных разумного размера не должно превышать времени работы сравнимых операций работы с моделями и журналами событий, реализованных в библиотеке

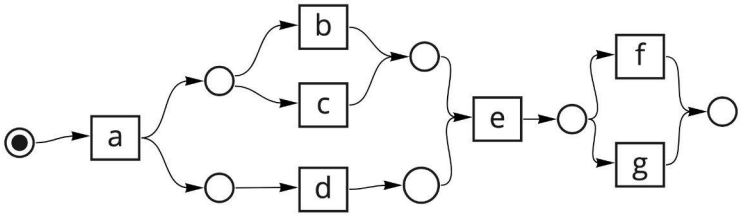
Описание метода

1. <u>Выделение “проблемных” вершин в сети</u>

Для этого использовался алгоритм построения alignments[1]. Неформально: по *следу* из лога строится *оптимальный* путь в модели, который примерно соответствует этому *следу*.

Предлагается выделять пары вершин в сети, которые будут соответствовать началу и концу “проблемных” участков.

Был предложен алгоритм выделения таких пар по alignments. Однако этот алгоритм не учитывает *шаги только в логе* из alignment (неясно, как именно их учитывать).



sync move

log-only move

log	a	b	>>	e	y	f
model	a	b	d	e	>>	f

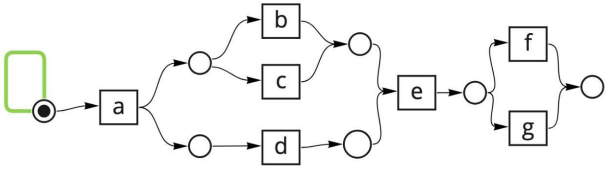
model-only move

Пример alignment для следа <a, b, e, y, f> для сети Петри слева

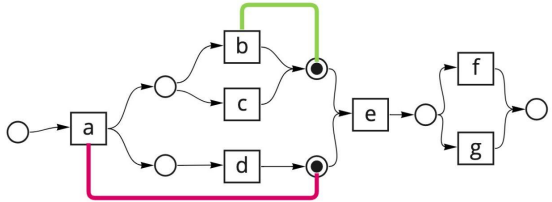
[1] *детально описан в книге [1]*

Описание метода

Пример выделения “плохих” пар вершин по alignment:
чёрным пунктиром отмечена выделенная “плохая” пара. Слева сеть, справа - alignment.

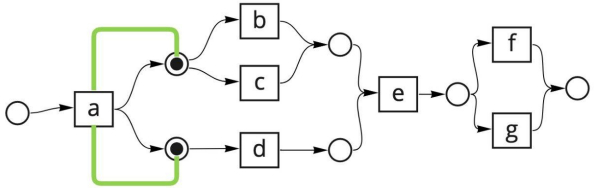


log	a	b	>>	e	f
model	a	b	d	e	f



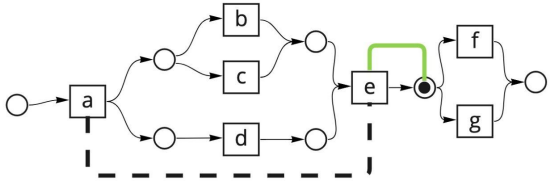
model-only move

log	a	b	>>	e	f
model	a	b	d	e	f



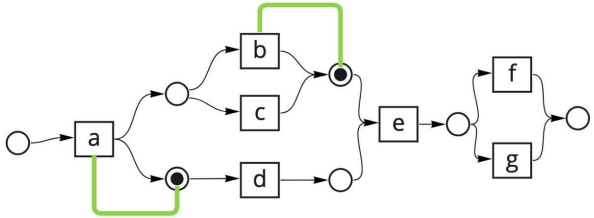
sync move

log	a	b	>>	e	f
model	a	b	d	e	f



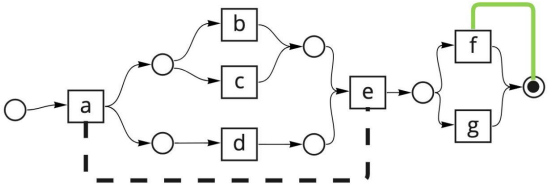
sync move

log	a	b	>>	e	f
model	a	b	d	e	f



sync move

log	a	b	>>	e	f
model	a	b	d	e	f



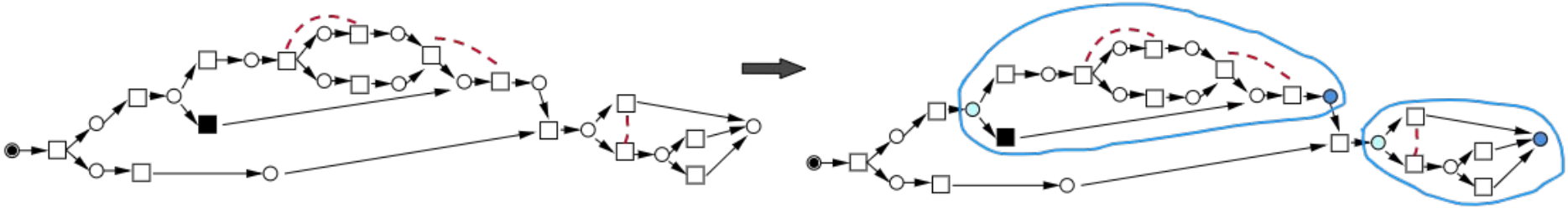
sync move

log	a	b	>>	e	f
model	a	b	d	e	f

Описание метода

2. Выделение гамаков

Находится минимальное по кол-ву включённых вершин мн-во непересекающихся гамаков в сети, такое, что гамаки покрывают все вершины из «плохих» пар, и для каждой такой пары обе вершины лежат в одном гамаке.



Пример выделения гамаков. “Плохие” пары отмечены красными пунктирами. Выделенные гамаки обведены.

Алгоритм выделения минимального гамака, содержащего выбранное мн-во вершин, не был мной найден (вообще, статей, использующих гамаки, не так много). Поэтому такой алгоритм с асимптотикой O(n + m), где n - кол-во вершин, m - кол-во рёбер, был предложен.

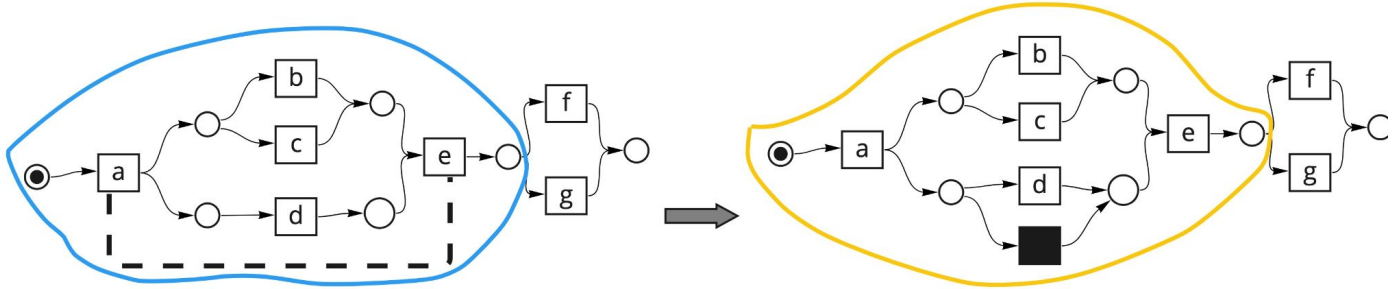
Затем, для нахождения мн-ва гамаков, которые содержат выделенные пары, был использован достаточно простой алгоритм.

Описание метода

3. Замена гамаков

Далее для каждого найденного гамака рассматривается мн-во действий внутри него. Из следов лога выделяются подпоследовательности действий из гамака. Затем по этой части лога с помощью некоторого алгоритма обнаружения процесса (был использован inductive miner, реализованный в PM4PY) строится сеть подпроцесса, которая “вставляется” в исправляемую сеть вместо гамака.

log trace	a	b	e	y	f



Пример замены выделенного гамака по логу. Часть следа <a, b, e> теперь воспроизводима в гамаке. Однако из-за игнорирование шагов только в логе след всё ещё не воспроизводим в сети.

Описание метода

4. Предварительное исправление модели

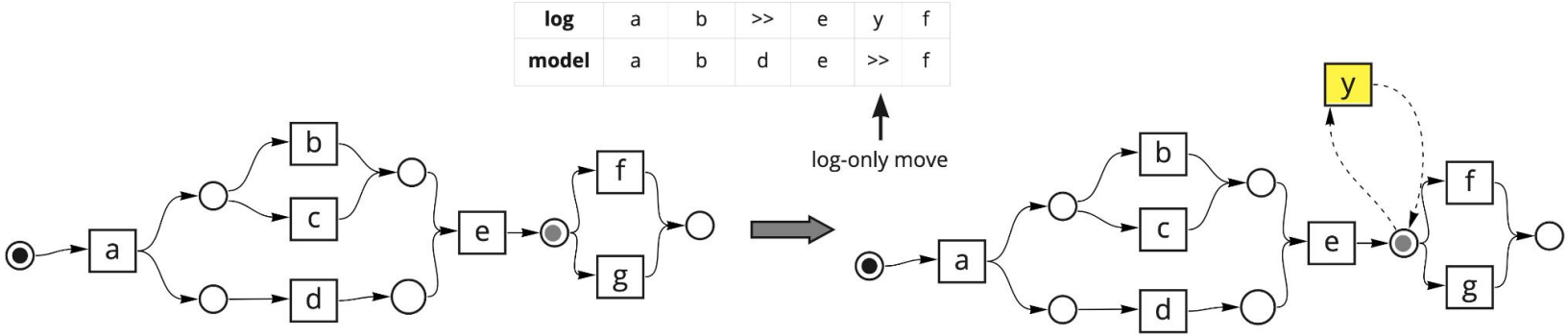
Описанный метод уже можно использовать для исправления сетей. Однако в нём совсем игнорируются шаги только в логе из alignments, что, конечно, не позволит добиться 100%-процентного соответствия следов из лога исправленной сети (100%-го значения fitness) в случае их наличия.

Можно:

1. изменить функцию стоимости при вычислении alignments так, чтобы минимизировать кол-во находимых шагов только в логе.
2. дополнительно исправить шаги только в логе перед применением общего алгоритма (это и буду делать)

В статье [4] предложен алгоритм исправления сети. В нём для исправления также используются alignments. Шаги только в модели исправляются достаточно просто, а для шагов только в логе используется нетривиальный метод. Для предварительного исправления сети в моём алгоритме используется базовая версия этого метода.

Описание метода



Пример предварительного исправления шага (y, >>) только в логе.

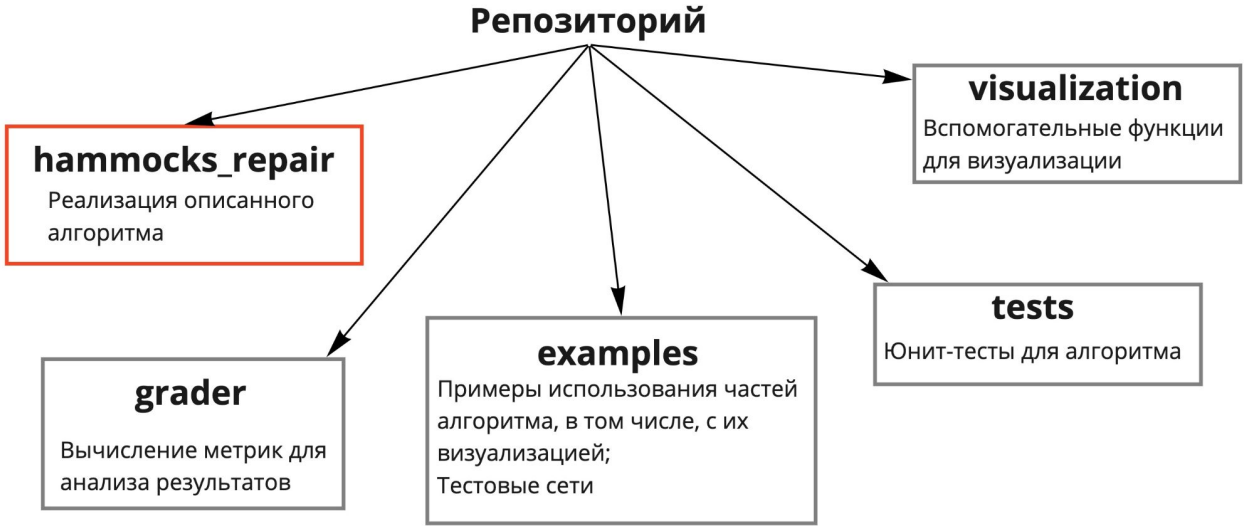
Описание метода

Общий метод

1. Нахождение alignments (в PM4PY уже реализовано)
2. Предварительное исправления сети по шагам только в логе
3. Нахождение “плохих” пар
4. Выделение гамаков, которые содержат “плохие” пары
5. Замена гамаков на исправленные подсети

Реализация

Все описанные алгоритмы реализованы на языке Python с использованием библиотеки PM4PY, исходный код размещён в Github-репозитории[1].



Структура репозитория

[1] *https://github.com/TrickmanOff/hammocks_repair_project*

Реализация

Каждая из упомянутых на схеме функций снабжена документацией в виде docstring.

В hammocks_repair я постарался организовать код приближённо к PM4PY (параметры, сигнатуры функций) для последующей возможности встраивания в библиотеку.

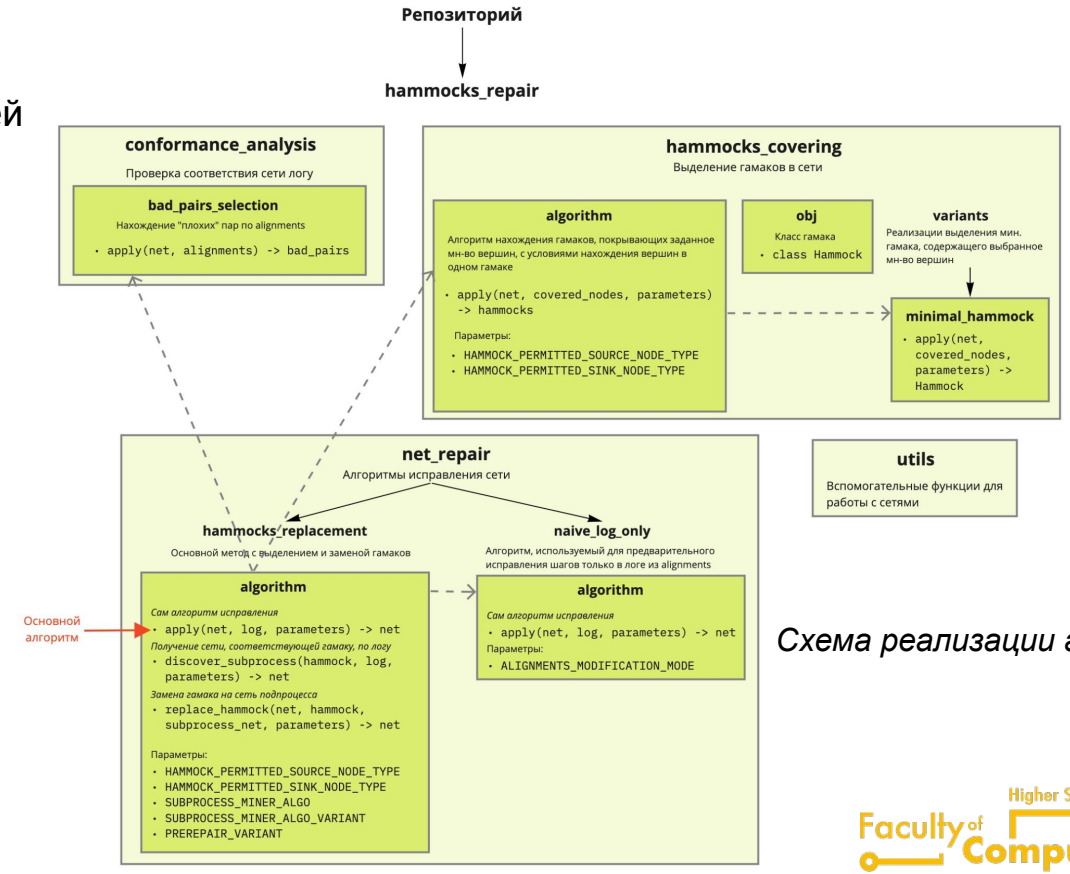


Схема реализации алгоритма

Демонстрация работы

> git clone https://github.com/TrickmanOff/hammocks_repair_project
> pip install ./hammocks_repair_project

базовый пример:
https://github.com/TrickmanOff/hammocks_repair_project/blob/main/demo/basic_example.py

пример с визуализацией промежуточных шагов и вычислением метрик:
https://github.com/TrickmanOff/hammocks_repair_project/blob/main/demo/all_in_one_example.py

Оценка результатов

Для оценки работы алгоритма были использованы распространённые метрики, которые приводятся во многих статьях с алгоритмами исправления модели:

- fitness: характеризует то, насколько лог воспроизводим в модели (соответствует recall в Data Science, если считать сеть классификатором следов по их воспроизводимости в ней)
  a) **fit_avg** (от 0 до 1): среднее значение “воспроизводимости” следов из лога
  b) **fit_tr** (от 0 до 1): доля полностью воспроизводимых следов из лога
  Ожидаемое значение: 1, кажется, что это основная задача алгоритма исправления сети
- precision**:** характеризует то, насколько модель допускает поведение, отличное от лога
  a) **prec** (от 0 до 1): значение precision, больше - лучше
- graph edit distance similarity**:** соответствует степени «похожести» сетей по как бы редакторскому расстоянию (1 – идентичные сети), описано в статье [5]
  a) **sim** (от 0 до 1): значение «похожести» исправленной сети и исходной исправляемой, вычисленное с помощью плагина [6]
  b) **sim*** (от 0 до 1): приближение значения «похожести» исправленной сети и исходной исправляемой, вычисленное с помощью алгоритма из NetworkX [7] (веса в формуле выставлялись мной в попытке отнормировать значение от 0 до 1, из-за чего можно будет видеть странные значения в таблице)
- footnotes similarity: соответствует степени “поведенчепохожести” сетей по отношению между парами действий в них
  a) **sim_f** (от 0 до 1): значение footnotes similarity исправленной сети и исходной исправляемой

Значения метрик, про которые явно не сказано другое, были вычислены встроенными функциями PM4PY.

Higher School of Economics Faculty of Computer science

Оценка результатов

Дополнительные значения:

- **size:** кол-во вершин в сети
- **time:** суммарное время работы алгоритма (сек.)
- **time_a**: время вычисления alignments (сек.)
- **time_p**: время предварительного исправления сети (сек.)
- **time_h**: время выполнения алгоритма выделения и замены гамаков (сек.)

Для тестирования были взяты 3 искусственно сгенерированные сети Петри: A, B и C (из [10]). Далее каждая из сетей была слегка точечно «испорчена». По исходной сети генерировался лог достаточно небольшого размера (150 следов), по которому далее исправлялась «испорченная» сеть.

Результат сравнивался с применения алгоритма обнаружения процесса по всему логу с помощью inductive miner (complete rediscovery).

Оценка результатов

		fit_{avg}	fit_{tr}	$prec$	sim	sim^*	sim_f	$size$	$time$	$time_a$	$time_p$	$time_h$
A1	initial	0.986	0.887	0.72	-	0.52	1	110	-	-	-	-
	hammocks	1	1	0.681	0.931	0.509	0.982	116	1.84	1.3	0.48	0.06
	rediscovery	1	1	0.056	0.503	0.394	0.021	77	0.68	-	-	-
A2	initial	0.948	0.653	0.714	-	0.504	1	109	-	-	-	-
	hammocks	1	1	0.67	0.899	0.484	0.968	116	1.79	1.26	0.48	0.05
	rediscovery	1	1	0.055	0.498	0.403	0.021	90	0.9	-	-	-
B1	initial	0.984	0.7	0.617	f	0.5	1	280	-	-	-	-
	hammocks	1	1	0.635	f	0.5	0.999	285	6.37	4.97	1.3	0.1
	rediscovery	1	1	0.492	f	0.462	0.971	236	0.69	-	-	-
B2	initial	0.925	0.46	0.668	f	0.504	1	276	-	-	-	-
	hammocks	1	1	0.615	f	0.504	0.997	284	6.39	4.91	1.34	0.14
	rediscovery	1	1	0.438	f	0.47	0.965	241	0.33	-	-	-
C1	initial	0.985	0.767	0.615	f	0.504	1	704		-	-	-
	hammocks	1	1	0.334	f	0.486	f	750	17.03	12.3	2.99	1.74
	rediscovery	1	1	0.093	f	0.476	f	741	6.67	-	-	-
C2	initial	0.985	0.693	0.581	f	0.501	1	706	-	-	-	-
	hammocks	1	1	0.287	f	0.486	f	747	29.22	25.8	2.67	0.75
	rediscovery	1	1	0.097	f	0.463	f	800	6.61	-	-	-

Результат применения алгоритма. Значение f в ячейке соответствует тому, что соответствующая метрика не была вычислена за разумное время

Выводы

- время работы непосредственно алгоритма исправления (без учёта нахождения alignments) соизмеримо с временем complete rediscovery
- алгоритм хорошо работает, если в сети есть небольшое число точечных несоответствий в некоторой отдалённости от стартовой и конечной вершин
- ожидаемо исправленная сеть больше “похожа” на исходную, чем при полном rediscovery
- однако у алгоритма возникают трудности, если, например, в качестве “плохой” была выделена пара вершин из параллельных веток (что, на самом деле, ожидаемо)

Алгоритм показывает вполне ожидаемые результаты, поэтому предложенное использование гамаков можно считать оправданным и имеющим свои преимущества для рассматриваемых случаев точечного несоответствия сети логу, и можно говорить о том, что алгоритм применим для исправления моделей.

План дальнейшей работы

Алгоритм:

Возможно, стоит ещё подумать над методом исправления шагов только в логе из alignments, потому что из-за него при выделении гамаков возникают трудности (выделяются большие гамаки или вся сеть).

Также хочется сравнить работу этого алгоритма с другими.

Апробация:

Я намерен оформить описанный алгоритм в виде научной статьи, чтобы её можно было где-то представить/опубликовать. Без этого, наверное, рано обсуждать включение реализации алгоритма в PM4PY. Написание статьи уже обсуждается с моим научным руководителем.

Без учёта отсутствия апробации алгоритма, реализация написана похожим на код из PM4PY образом, поэтому ожидается, что её легко будет встроить в библиотеку. Возможно, придётся внести какие-то незначительные правки, написать более подробную документацию и составить покрывающие большее число случаев тесты.

Список использованных источников

1 - Wil van der Aalst. *Process Mining: Data Science in Action*. Springer, 2016.
2 - Fubo Zhang и E. H. D'Hollander. «*Using hammock graphs to structure programs*»
3 - Berti, A., van Zelst, S.J., van der Aalst, W.M.P. (2019): *Process Mining for Python* (PM4Py)
4 - Dirk Fahland и Wil M. P. van der Aalst. «*Model repair — aligning process models to reality*»
5 - Remco Dijkman и др. «*Similarity of business process models: Metrics and evaluation*»
6 - B.F.A Hompes. «*On Decomposed Process Discovery: How to Solve a Jigsaw Puzzle with Friends*»
7 - *NetworkX* - Python library for studying graphs and networks. https://networkx.org/.

Faculty of Computer science
Higher School of Economics

computerscience@hse.ru
https://cs.hse.ru/en/
facebook.com/hsecs