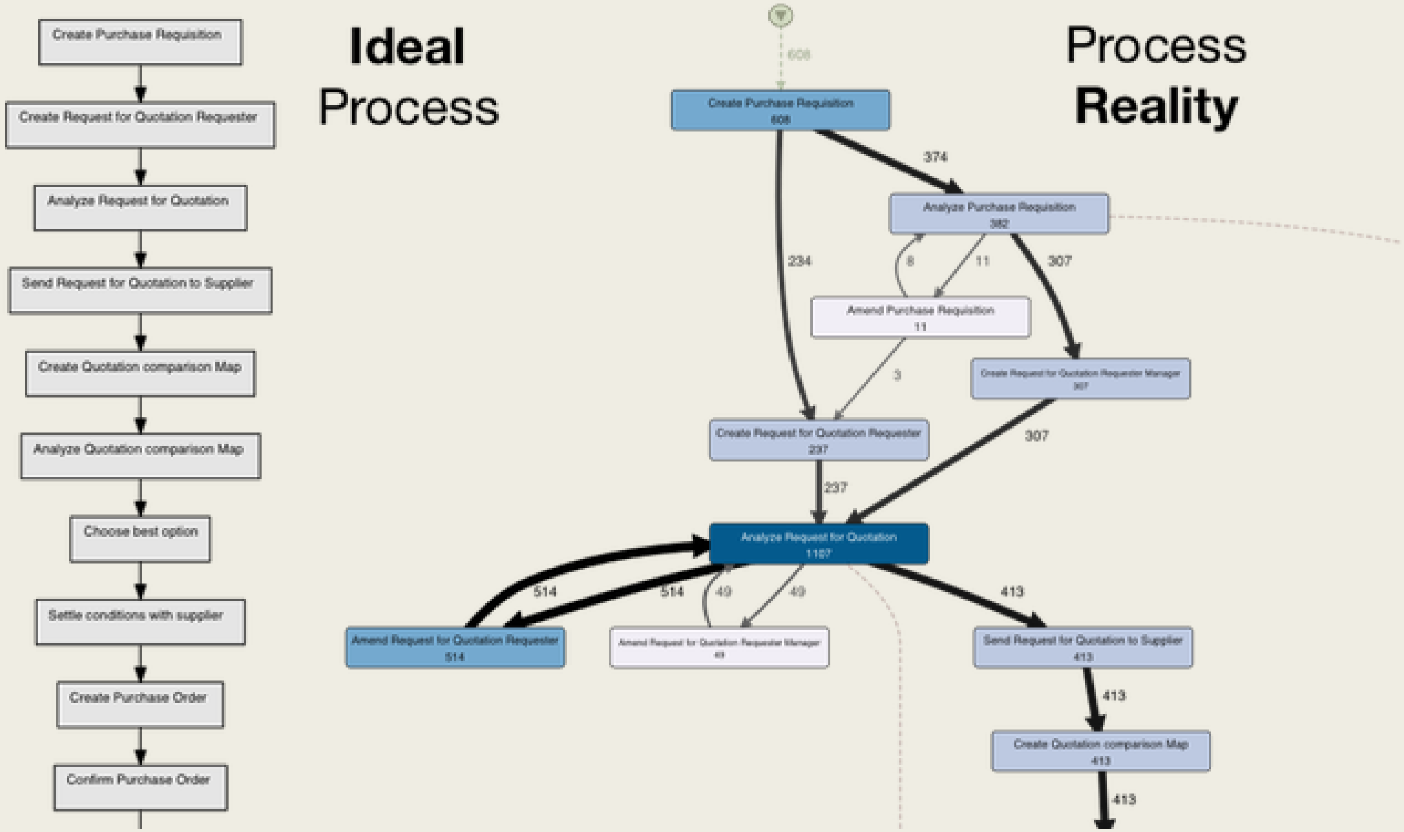
МЕТОД ЭФФЕКТИВНОЙ ДЕКОМПОЗИЦИИ МОДЕЛЕЙ ПРОЦЕССОВ ДЛЯ ИХ ПОЧИНКИ

Выполнил:
Тихонов Семён
стажёр-исследователь НУЛ ПОИС,
студент группы БПИ162.

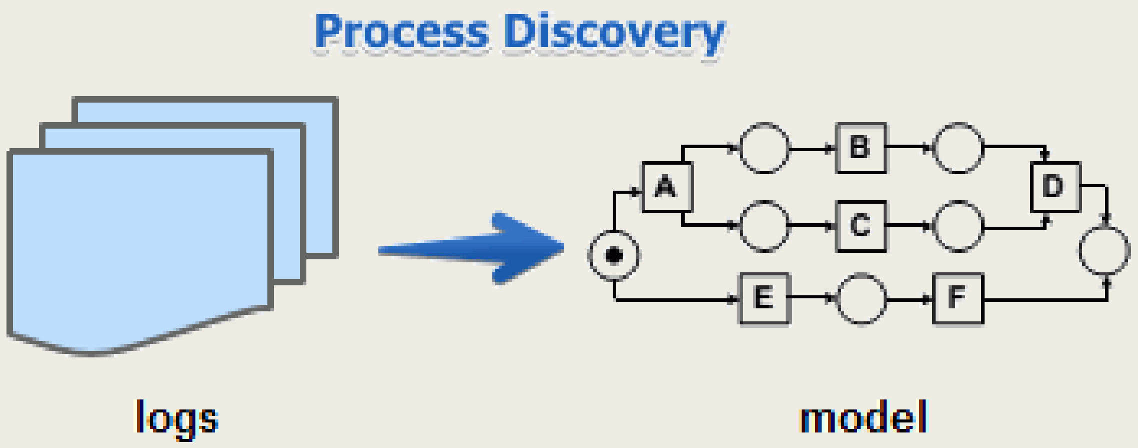
Научный руководитель:
Мицюк Алексей Александрович.

Process Mining

- Цель: усовершенствование процессов на основании изучения журналов событий.

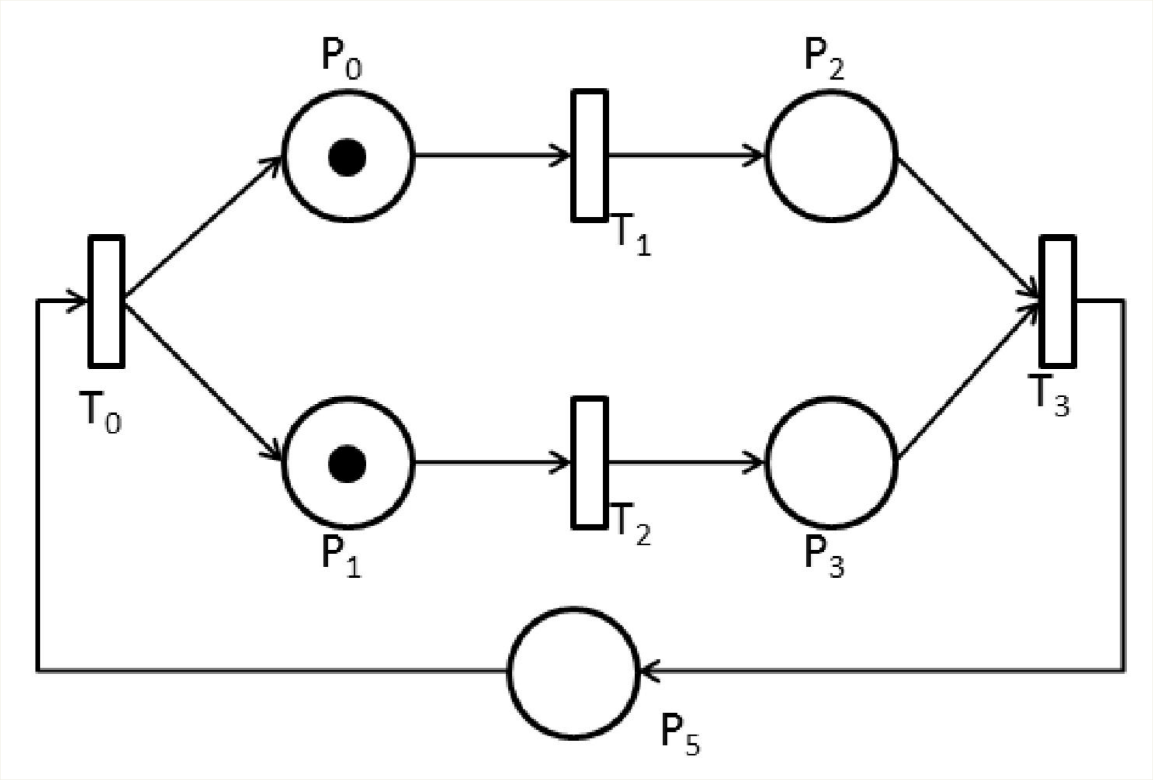


Модели процессов



Сеть Петри

- Сеть Петри - тройка $N = (P, T, F)$
- P и T – конечные множества состояний и переходов
- $F \subset (P \times T) \cup (T \times P)$ – множество дуг



Починка моделей процессов

Проблема: модели процессов часто ***не соответствуют реальному поведению*** системы.

Подход: создать принципиально новую модель на основе существующих логов, используя discovery-алгоритмы.

Недостатки подхода:

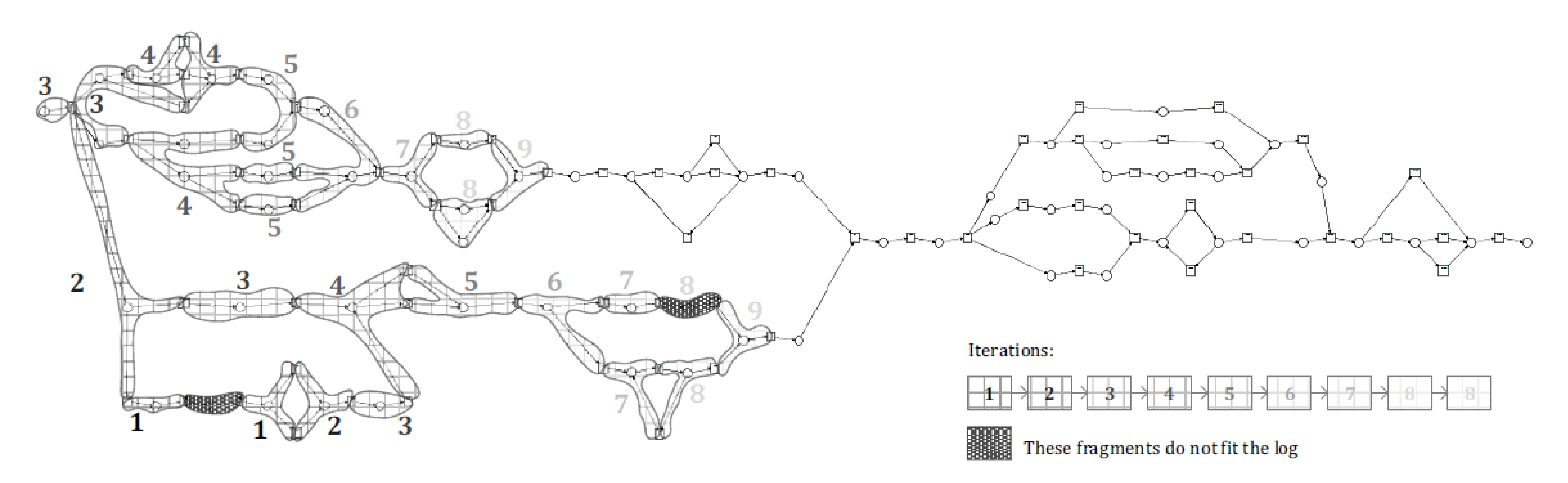
- Уже построенная модель могла иметь высокую ценность (например, у нее была хорошая структура или она была создана экспертами, которым впоследствии предстоит продолжать работу с этой моделью).

Основные принципы починки

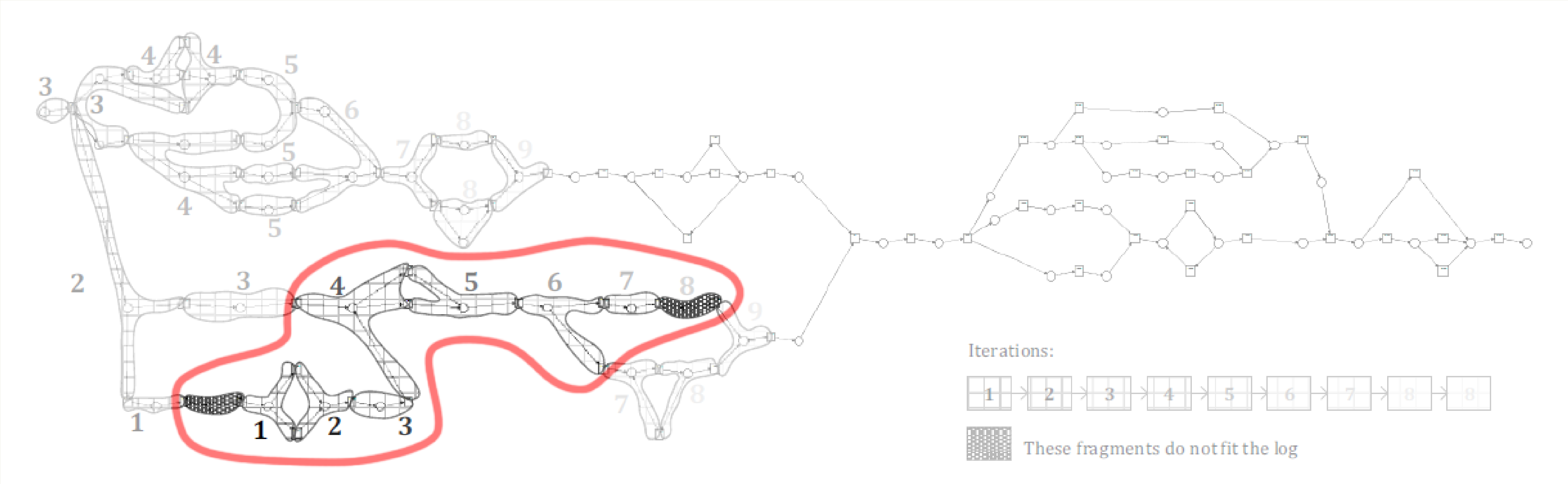
- Не создавать принципиально новую модель
- **Исправлять** уже существующую модель
- **Изменять как можно меньше** фрагментов существующей модели
- Изменять фрагменты модели, не соответствующие сублогу.

Жадный алгоритм починки моделей

- **Подход:** чинить большой фрагмент модели, включающий в себя все проблемные части декомпозиции.



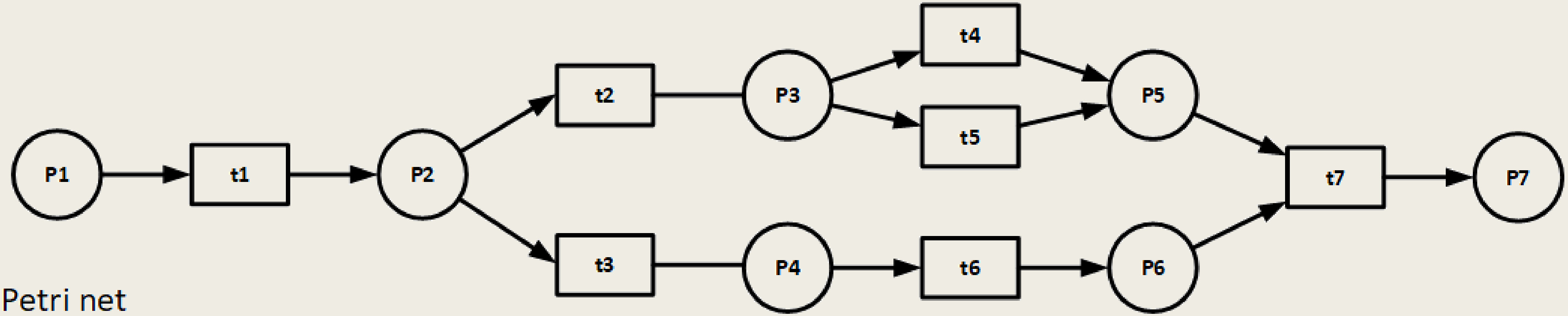
Как добиться того, чтобы не выделялись лишние части?



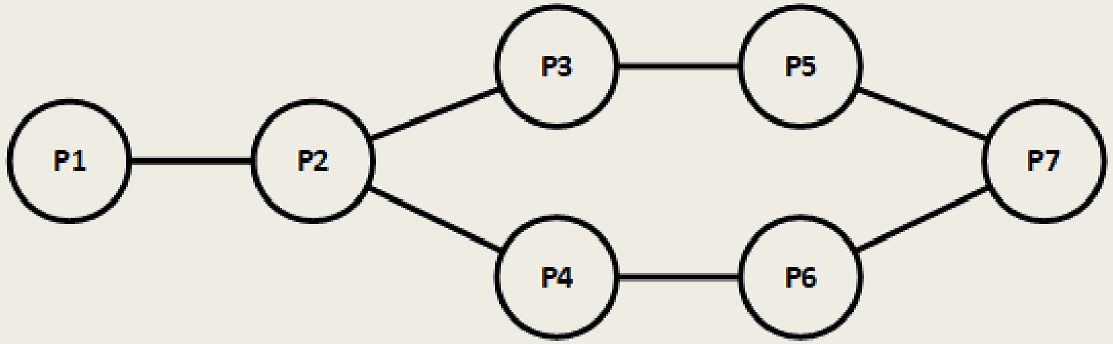
Граф декомпозиции

- Для данной сети Петри $N = (P, T, F)$ граф декомпозиции – пара (V, E), где $V = P$ – множество всех переходов в сети Петри, а $E = \{p_1 p_2 \mid p_1, p_2 \in P, \exists\, t \in T: (F(p_1, t) \neq 0) \land (F(t, p_2) \neq 0)\}$ – множество дуг.
- ГД – способ показать связь между двумя состояниями в сети Петри вне зависимости от количества переходов между ними.
- Существование дуги в ГД между вершинами p_1 и p_2 означает, что в изначальной модели состояния p_1 и p_2 непосредственно соединены хотя бы одним переходом.

Граф декомпозиции



a) Petri net

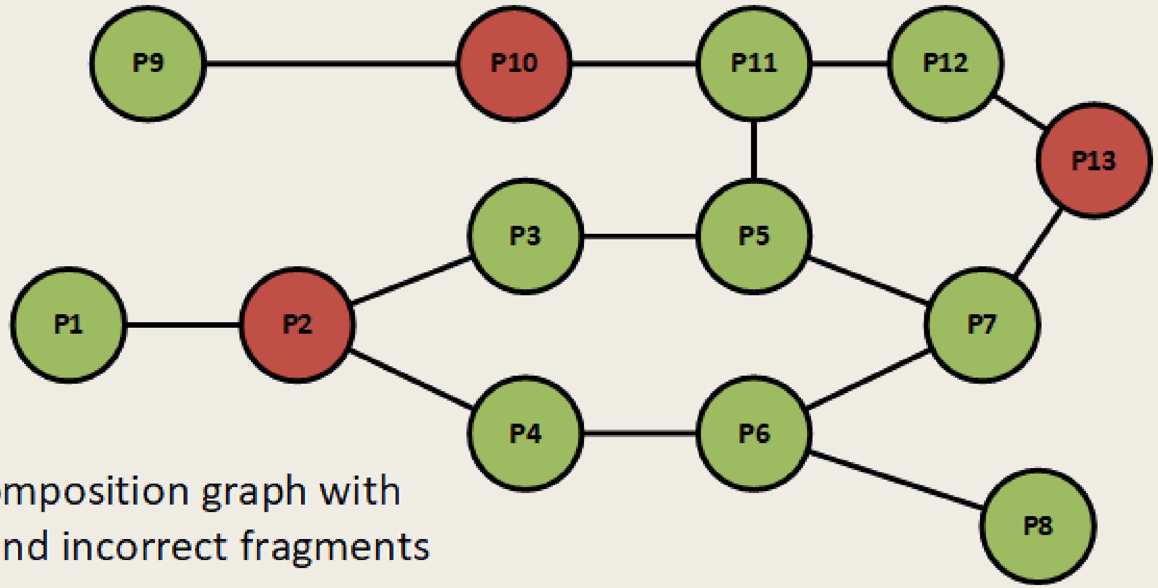


b) Decomposition graph

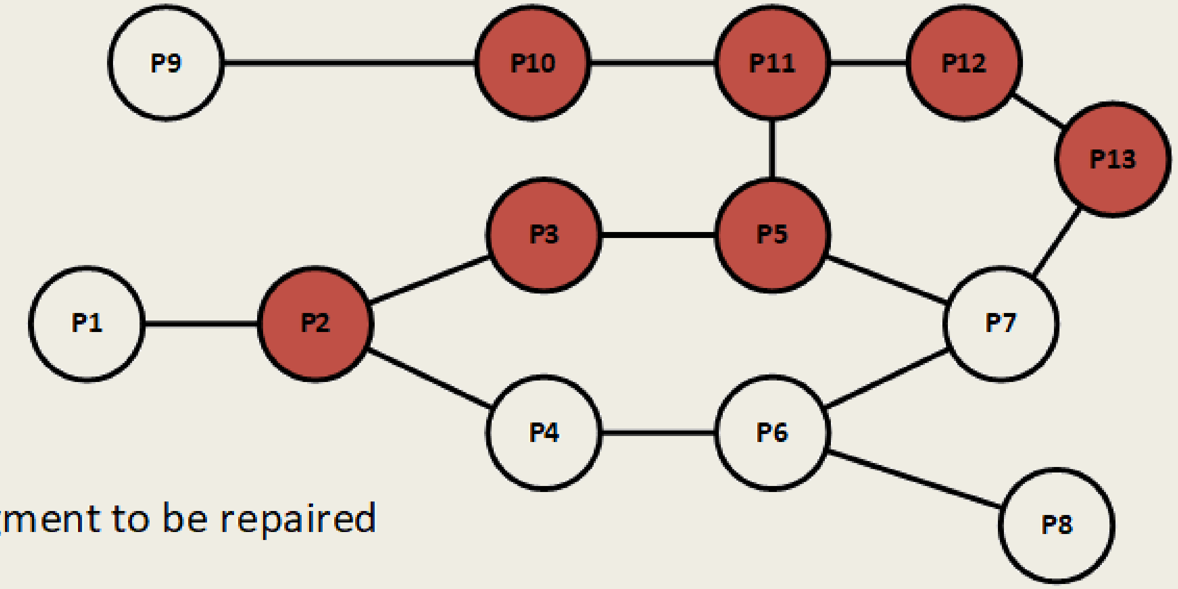
Постановка задачи

- **Имеем:** неориентированный граф с вершинами двух типов: **красные** (некорректные фрагменты модели) и **зелёные** (корректные фрагменты)
- **Цель:** построить подграф, который соединит между собой все красные вершины, при этом в подграф будет включено наименьшее возможное число зеленых вершин.

Постановка задачи



a) decomposition graph with
correct and incorrect fragments



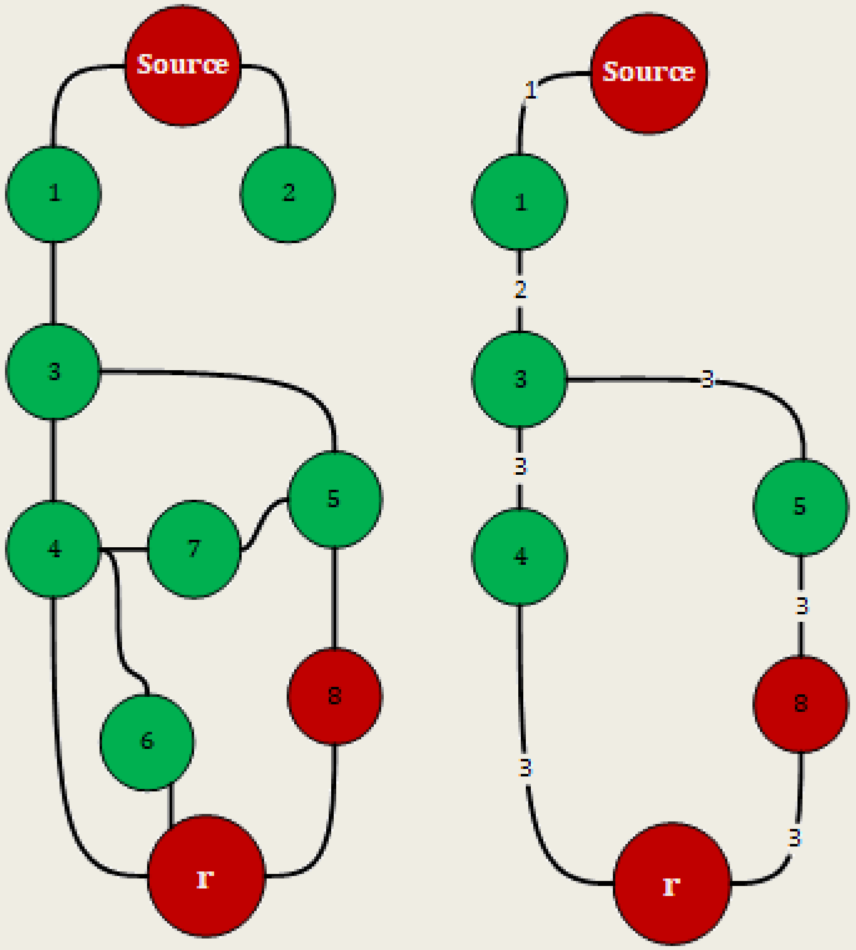
b) fragment to be repaired

Доминаторы в теории графов

- **Определение:** Пусть S – входной узел (вершина) графа. Тогда считается, что *узел d доминирует над узлом n*, если любой путь от входного узла графа к n проходит через d. В частности, каждый узел доминирует над самим собой. Обозначение: $d\ dom\ n$

Функция getAllShortestPaths

- Находит множество всех путей минимальной длины из вершины *Source* в вершину *r* с помощью стандартного поиска в глубину.
- Стоимость перехода по дуге, заканчивающейся в вершине красного цвета, равна нулю; стоимость перехода по дуге, заканчивающейся вершиной зеленого цвета, равна единице.
- Возвращает объединение всех найденных путей.



a) paths of minimum length from *Source* to *r* are to be found

b) the unification of paths *Source-1-3-4-r and Source-1-3-5-8-r*

Алгоритм

- B – множество вершин графа декомпозиции с идеальным соответствием суб-логу.
  R – множество вершин графа декомпозиции без идеального соответствия суб-логу
- Входные данные: $G = (V, E)$ – граф декомпозиции, где V – пара (B, R).
- Выходные данные: граф $G_{fitting}$ и граф $G_{unfitting}$.

Алгоритм

1. Случайным образом выбрать вершину $Source$ из множества вершин R.
2. Создать пустое множество $Doms$, которое будет содержать доминаторов, найденных в шаге *3.a)*.
3. Для каждой вершины $r \in R$:
   a) Создать граф кратчайших путей G_r с помощью функции $getAllShortestPaths(G, Source, r)$;
   b) В G_r *найти всех доминаторов над вершиной* r;
   c) Добавить найденные доминаторы ко множеству $Doms$*.*

Алгоритм

4. Для всех $d_1, d_2 \in Doms$, таких что дуга $d_1 d_2 \in E$:
   a) Добавить d_1 и d_2 к множеству $Vertices_{unfitting}$;
   b) *Добавить* дугу $d_1 d_2$ к множеству $Edges_{unfitting}$.
5. Для каждой вершины $r : (r \in R) \wedge (r \notin Vertices_{unfitting})$:
   a) Найти $P = (V_{path}, E_{path})$ – кратчайший путь из вершины r к ближайшей вершине из множества $Vertices_{unfitting}$;
   b) Добавить все вершины $v_p \in V_{path}$ к множеству $Vertices_{unfitting}$;
   c) Добавить все дуги $e_p \in E_{path}$ к множеству $Edges_{unfitting}$;

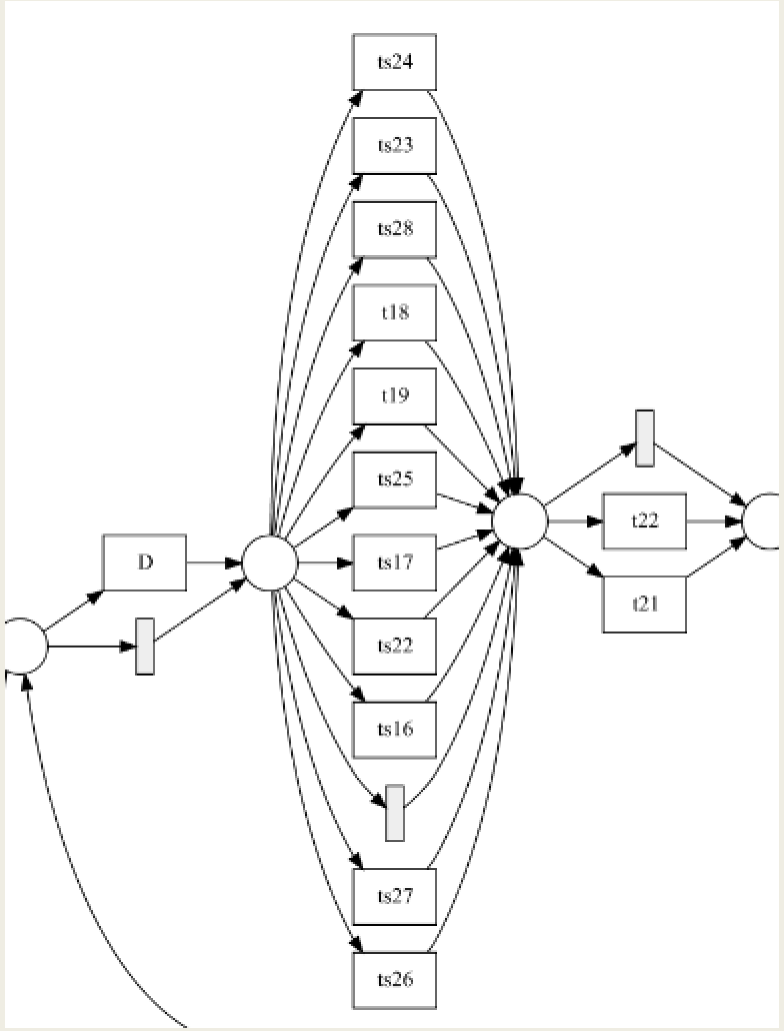
Алгоритм

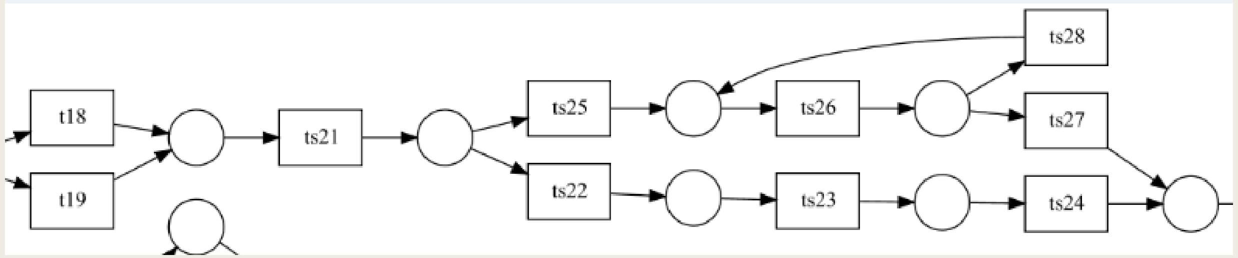
6. Создать множества $Vertices_{fitting} = \{v \mid v \in V \setminus Vertices_{unfitting}\}$ и $Edges_{fitting} = \{e \mid e \in E \setminus Edges_{unfitting}\}$.
7. Вернуть граф $G_{fitting} = (Vertices_{fitting}, Edges_{fitting})$ и граф $G_{unfitting} = (Vertices_{unfitting}, Edges_{unfitting})$.

Оценка результатов

	Smart	Greedy	Smart	Greedy	Smart	Greedy	Smart	Greedy
	Size of wrapper		Iterations		Size of final model		time	
1	47	113	1	5	142	170	242862	2692
2	55	126	1	12	152	171	225070	1336
3	50	125	1	11	135	167	6262	1248
4	80	133	1	15	174	166	135474	1058
5	60	123	1	8	175	168	162256	932
6	48	120	1	9	135	168	4892	1254
7	31	45	1	1	142	149	16264	452
8	53	117	1	9	144	167	13368	906
9	38	61	1	4	143	154	69310	456
10	62	123	1	9	170	170	578948	1084
average	*52,4*	*108,6*	*1*	*8,3*	*151,2*	*165*	*145470,6*	*1141,8*
median	*51,5*	*121,5*	*1*	*9*	*143,5*	*167,5*	*102392*	*1071*
max	*80*	*133*	*1*	*15*	*175*	*171*	*578948*	*2692*
min	*31*	*45*	*1*	*1*	*135*	*149*	*4892*	*452*

Сравнение разработанного и жадного методов





Спасибо за внимание!

Q&A time