

# Координация пропуска вагонопотока

команда: “Люди работают. Понимать надо”,  
состав: Павел Снопов, Герман Магай, Сергей Ермоленко

# Команда “Люди работают. Понимать надо”, наши роли:

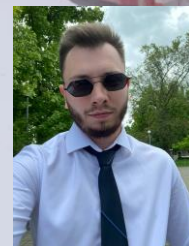
Павел Снопов, ИППИ РАН - реализация решения и разработка



Герман Магай, ВШЭ - математическая модель и реализация

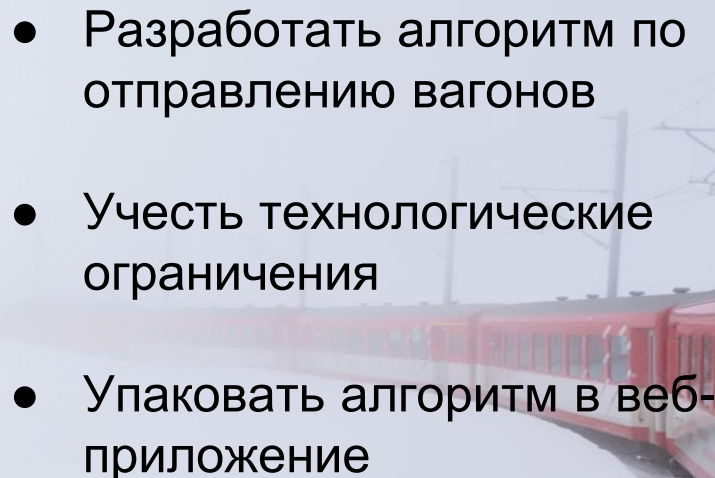


Сергей Ермоленко, ВГУ - реализация веб-приложения



# Формулировка задачи



- 
- Разработать алгоритм по отправлению вагонов
  - Учесть технологические ограничения
  - Упаковать алгоритм в веб-приложение



# Математическая модель задачи:

Задача относится к типу транспортных задач и формулируется, как проблема целочисленного линейного программирования

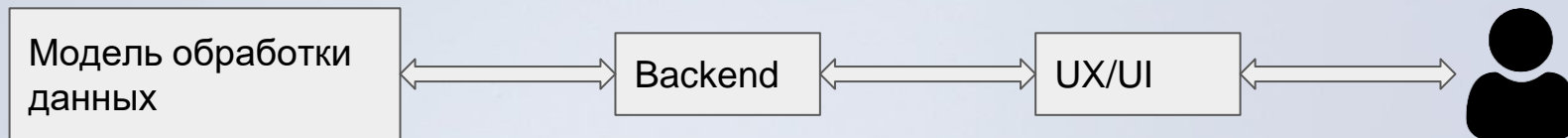
Подходы к решениям:

- Метод декомпозиции Данцига — Вулфа
- Метод северо-западного угла

Статьи, в которых предложено решения более общих задач для других стран:

- R. Fukasawa et. al. Solving the Freight Car Flow Problem to Optimality
- K. Holmberg et. al. Improved Empty Freight Car Distribution

# Структура решения:



Мы предлагаем решение на основе метода северо-западного угла. Алгоритм упакован в веб-приложение, информирующее работников об отправлении вагонов в составе поездов.

К преимуществам нашего решения можно отнести:

- масштабируемость
- интерпретируемость
- скорость работы
- удовлетворение технологическим ограничениям
- возможность улучшения алгоритма с помощью нейронных сетей

# Формулировка итерационного решения

1. Сортируем поезда по длине их маршрутов.
2. Берем самый первый поезд и выписываем все пары городов, которые он проезжает.
3. Выбираем самую длинную пару городов.
4. Между этими городами отправляем число вагонов, равное минимуму из пропускной способности на этом участке и потребности в отправке
5. Переписываем матрицу отправляемых вагонов и пропускную способность на выбранном участке
6. Так делаем, пока для всех пар городов данного поезда не будет записано, сколько вагонов отправляется.



## Стек технологий:

1. Python



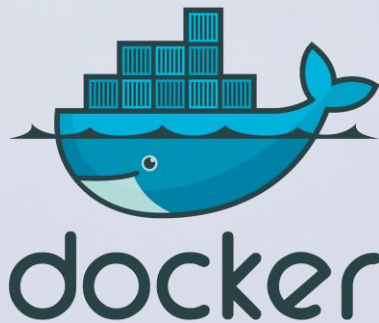
1. Dash



1. Pandas



1. Docker



# Программный код:

In [401...

```
for i, train in enumerate(trains_codes):
    train_route = routes[i]
    route_len = len(train_route)
    train_free_carriage = np.array(free_carriages[i])
    Cars[i] = {}
    for j, start in enumerate(train_route):
        for end in train_route[j:-1]:
            start_ind = train_route.index(start)
            end_ind = train_route.index(end)
            path = (start, end)
            need = Needs[path[0]-1, path[1]-1]
            train_capacity = min(train_free_carriage[start_ind, need], train_capacity)
            cars = min(need, train_capacity)
            Cars[i][path] = cars
            Needs[path[0]-1, path[1]-1] -= cars
            train_free_carriage[start_ind : end_ind] -= cars
```

```
def update_drop(station):
    if station:
        return f'Вы сотрудник станции {station}'
    else:
        return 'Выберите станцию для расписания вагонопотока'

@app.callback(
    Output('table', 'data'),
    Output('table', 'columns'),
    State('input1', 'value'),
    State('drop1', 'value'),
    Input('button', 'n_clicks')
)

def update_table(ecz, city, click):
    if click>0:
        df = problem.get_solution_for_city((ecz-1), city)
        df = df.rename(columns={'trains': 'Поезд', 'number of cars': 'Количество вагонов для сцепки',
                                'arrival time': 'Время прибытия', 'departure time': 'Время отбытия'}, inplace=False)

        df1 = df.to_dict('records')
        columns = df.columns
        columns=[{"name": i, "id": i} for i in columns]

        return df1, columns
    else:
        return [], []
```

Программный код: <https://github.com/Snopoff/RZD-Car-Flow/tree/main>



## Предусмотрены технологические ограничения:

1. Маршруты не могут быть циклами и иметь возвраты
2. Минимизация перецепления вагонов
3. Минимизация времени, потраченного на маршрут для каждого вагона
4. Минимизация простоя
5. Отправляем вагонов не больше, чем надо
6. Отправляем вагонов не больше, чем можем

# Результат работы алгоритма:

Матрица отправляемых вагонов

Города	Еманжелинск (7)	9	3	23	25	32	37	0
	Полетаево (6)	21	27	24	9	1	0	3
	Челябинск (5)	38	38	1	28	0	33	14
	Муслюмово (4)	12	39	1	0	2	14	20
	Миасс (3)	15	5	0	27	16	31	24
	Кыштым (2)	26	0	7	34	20	27	35
	Златоуст (1)	0	38	38	25	29	7	10
		1	2	3	4	5	6	7
		Города						

Матрица оставшихся вагонов

Города	7	0	0	7	0	0	8	0
	6	4	3	0	4	0	0	0
	5	0	6	0	0	0	0	0
	4	12	0	0	0	0	0	0
	3	0	0	0	0	0	6	19
	2	5	0	0	11	0	0	0
	1	0	7	0	9	8	5	0
		1	2	3	4	5	6	7
		Города						

# Пользовательский интерфейс:

До запуска алгоритма

**Координация пропуска вагонопотока**

Выберите экземпляр вагонопотока

Выберите станцию  
Select...

Выберите станцию для расписания вагонопотока

**Выполненные расчеты**

После запуска алгоритма

**Координация пропуска вагонопотока**

Выберите экземпляр вагонопотока

Выберите станцию  
Кыштым

Вы сотрудник станции Кыштым

**Выполненные расчеты**

Поезд	Количество вагонов для сцепки	Время прибытия	Время отбытия
853	0	13:35	13:58
881	0	14:30	15:39
743	8	01:00	02:01
399	12	02:00	02:33
930	22	01:48	02:10
658	20	01:31	03:01
332	9	02:51	03:28
617	26	01:42	03:03
967	0	10:25	10:42
692	1	11:23	12:07

UI интерфейс был создан для работников станций РЖД для своевременного обслуживания вагонопотока