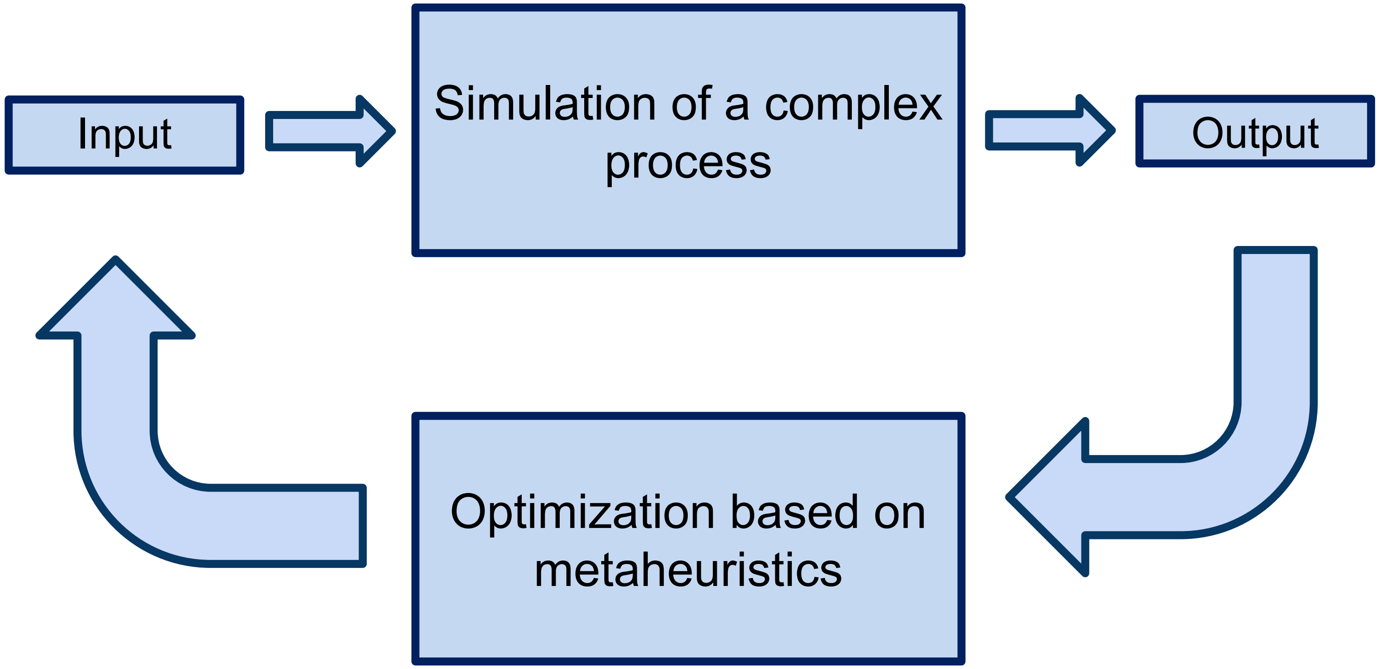
Metaheuristics in optimization of complex processes

Paweł Gora
Faculty of Mathematics, Informatics and Mechanics
University of Warsaw

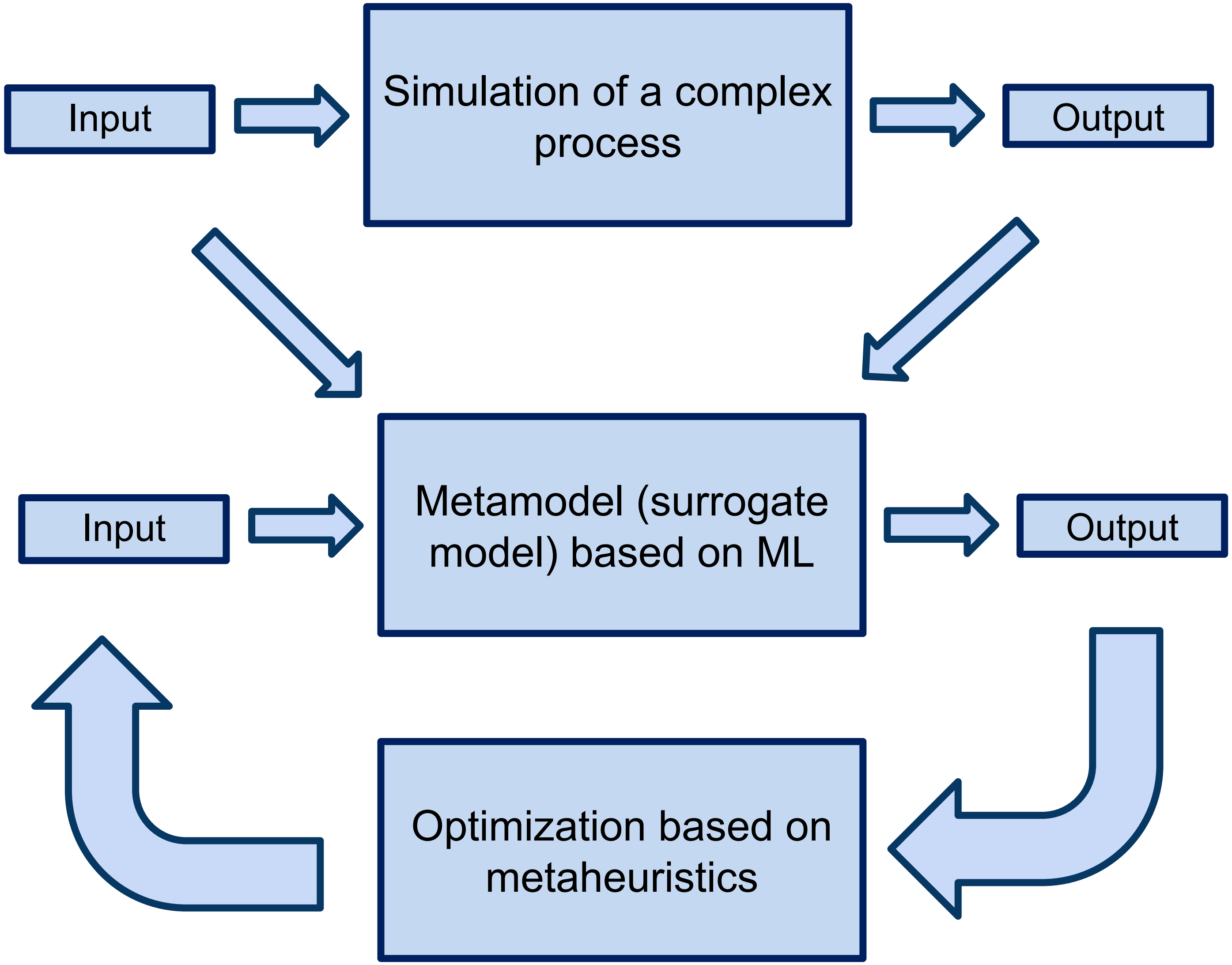
Group of Logic Research Seminar

27.11.2020

The general schema


Input
Simulation of a complex process
Output
Optimization based on metaheuristics

The general schema


Input
Simulation of a complex process
Output
Input
Metamodel (surrogate model) based on ML
Output
Optimization based on metaheuristics

Prediction “What-if”

What will happen if we change something in the road network?
What will happen if we change traffic signal settings?

$\Downarrow$

Step toward traffic optimization

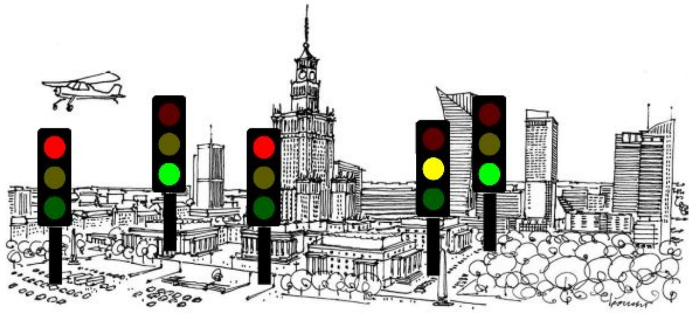
Prediction “What-if”

➔ The most popular and important tools to analyze traffic: **traffic simulations**.

➔ But accurate traffic simulations are time-consuming, especially in a large scale.

➔ In most of the optimization problems in transport, we have to use metaheuristics to find (heuristically) optimal solutions (these problems are often NP-hard). In such cases we have to run large number of such simulations, with different settings.

Nr of games: 10^{700}



Nr of settings: $> 120^{800}$
(and the problem is proved to be NP-hard!)

More than the number of atoms in the visible Universe!

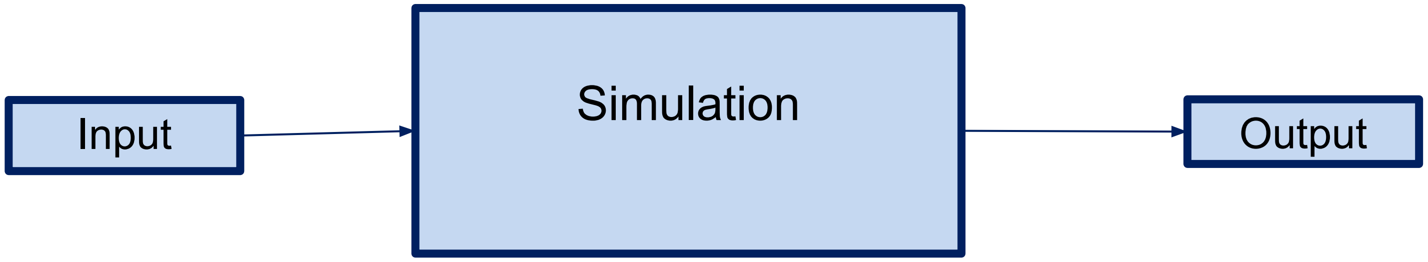
Prediction “What-if”

- ➔ The most popular and important tools to analyze traffic: **traffic simulations**.
- ➔ But accurate traffic simulations are time-consuming, especially in a large scale.
- ➔ In most of the optimization problems in transport, we have to use metaheuristics to find (heuristically) optimal solutions (these problems are often NP-hard). In such cases we have to run large number of such simulations, with different settings.
- ➔ We would like to do it as efficiently as possible.
- ➔ We can distribute computations on GPU or many machines, but it may be expensive and has limitations.

Prediction “What-if”

- ➔ The most popular and important tools to analyze traffic: **traffic simulations**.
- ➔ But accurate traffic simulations are time-consuming, especially in a large scale.
- ➔ In most of the optimization problems in transport, we have to use metaheuristics to find (heuristically) optimal solutions (these problems are often NP-hard). In such cases we have to run large number of such simulations, with different settings.
- ➔ We would like to do it as efficiently as possible.
- ➔ We can distribute computations on GPU or many machines, but it may be expensive and has limitations.
- ➔ In many cases we are not interested in the simulation process, but <u>only</u> in its outcomes. So, maybe we can somehow approximate the outcome based on partial output data?

Problem



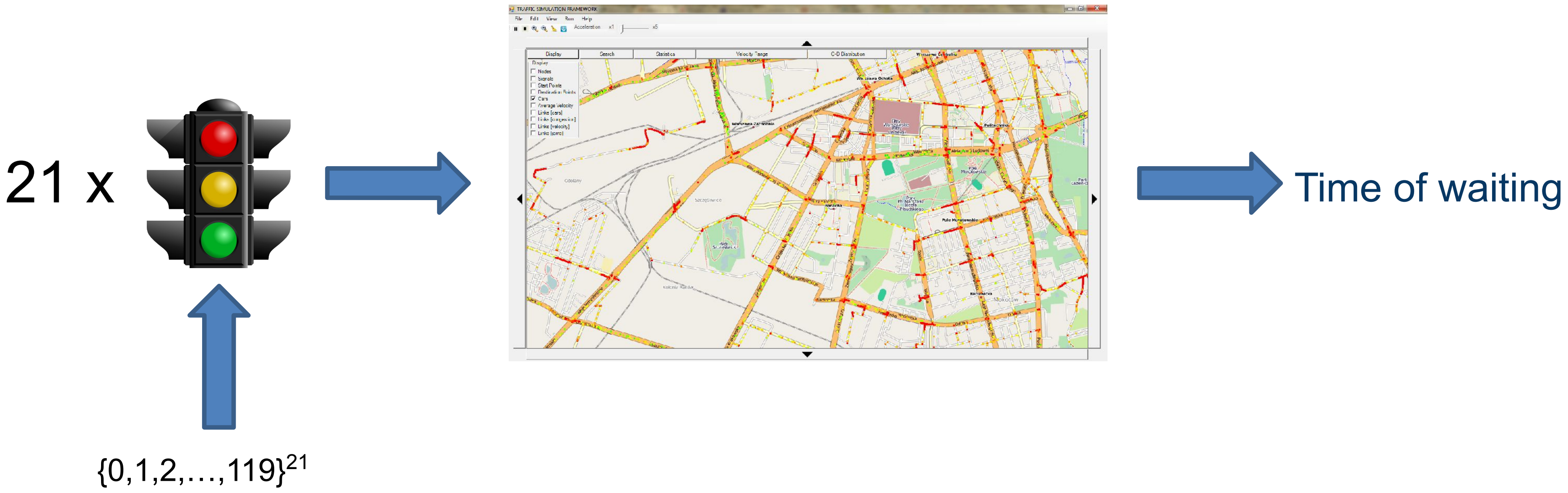
F: X -> Y

Example: X – traffic signal setting, Y – total delay, total time of waiting etc (real number)
(Y can be also a random variable in case of stochastic models)

Can we „compute” **F** faster / easier than by running traffic simulations?
Can we find a „metamodel” (surrogate model) approximating outcomes of simulations?

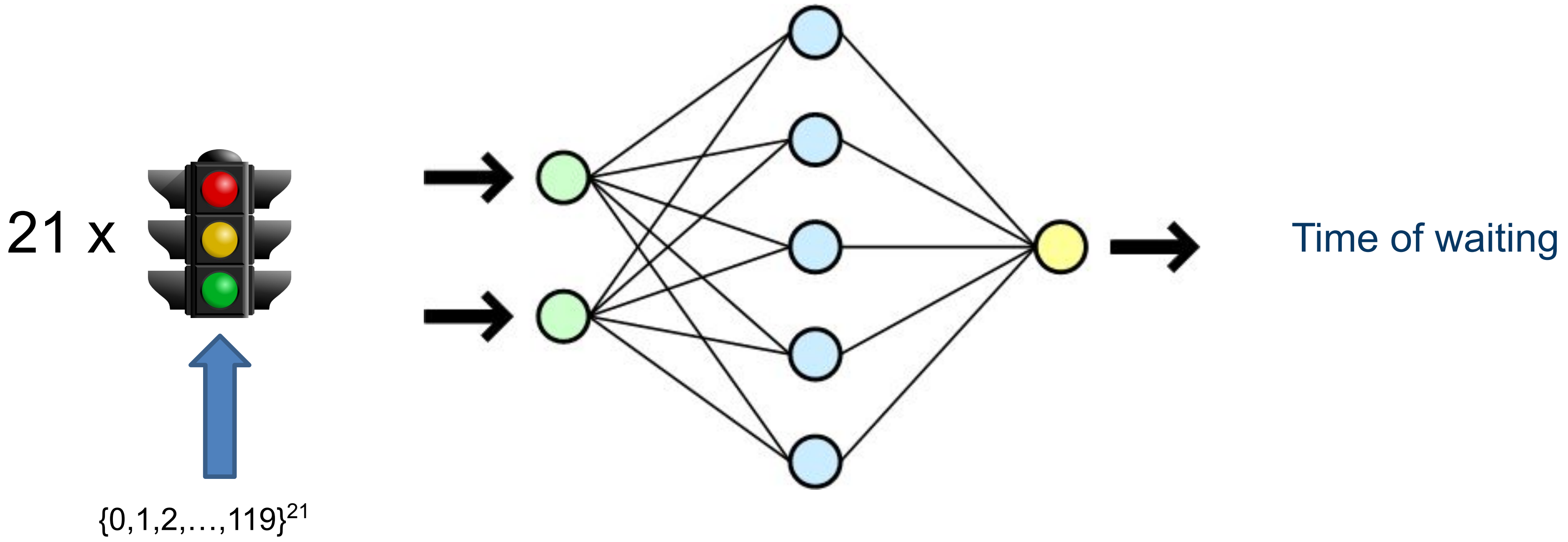
Solution

Using Traffic Simulation Framework we generated set composed of 105336 elements, divided it into training set (85336 elements) and test set (20000 elements). Each run simulated 10 minutes of traffic with 42 000 cars on a realistic map of Warsaw (OSM).



Solution

We focused on approximating the total waiting times on a red signal as a function of signal offset settings (signal offset setting = offsets of 21 traffic signal representatives on a Stara Ochota district in Warsaw).



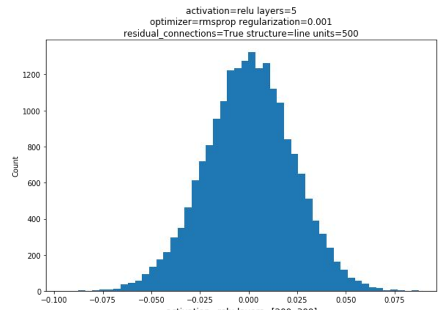
Solution

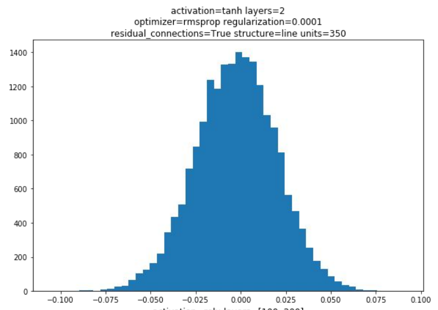
TensorFlow

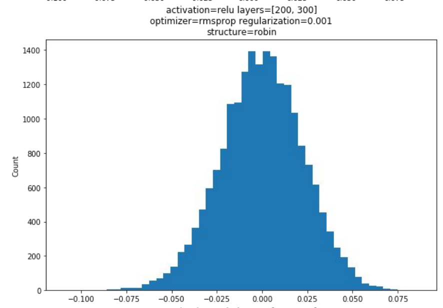
➔ We developed a **TensorTraffic** tool for approximating outcomes of traffic simulations using NN and predicting what may happen if we change traffic signal settings.

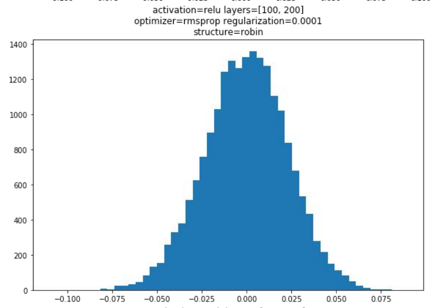
➔ Initially, we tested only feed forward neural networks. We found out, that, indeed, **outcomes of traffic simulation can be approximated using NN** with a good accuracy (best mean error on a test set: ~1.62%, maximal error: ~10.95%).

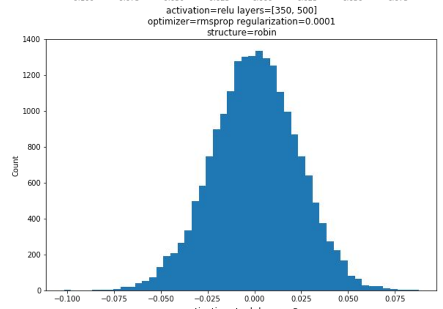
Distribution of error on a test set

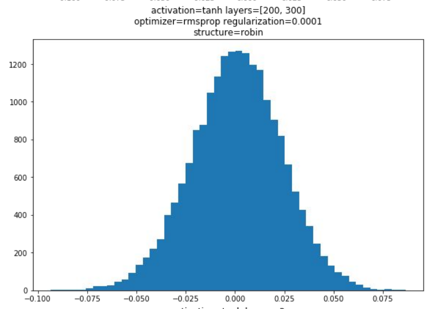


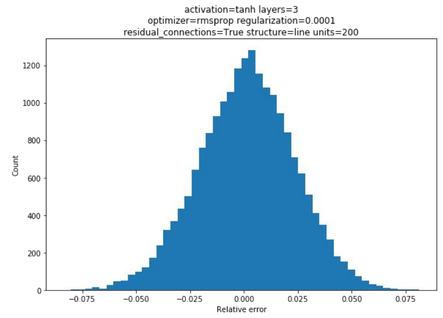


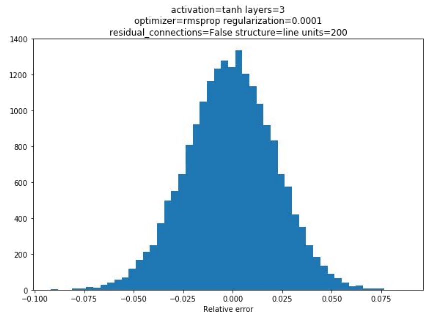












Solution

TensorFlow

- ➔ We developed a **TensorTraffic** tool for approximating outcomes of traffic simulations using NN and predicting what may happen if we change traffic signal settings.

- ➔ Initially, we tested only feed forward neural networks. We found out, that, indeed, **outcomes of traffic simulation can be approximated using NN** with a good accuracy (best mean error on a test set: ~1.62%, maximal error: ~10.95%).

- ➔ Time of simulating 10 minutes of traffic in a large-scale (e.g., Warsaw) using a microscopic model (TSF) – **(~30 seconds on standard machines)**.

- ➔ Time of inferencing neural network: **~0.8 ms** (time of training with GPU: ~10-15 minutes).

- ➔ And in the investigated case we don't even need large networks (3-4 layers with a few hundred neurons are sufficient).

Solution

➔ We developed a **TensorTraffic** tool for approximating outcomes of traffic simulations using NN and predicting what may happen if we change traffic signal settings.

➔ Initially, we tested only feed forward neural networks. We found out, that, indeed, **outcomes of traffic simulation can be approximated using NN** with a good accuracy (best mean error on a test set: ~1.62%, maximal error: ~10.95%).

➔ Time of simulating 10 minutes of traffic in a large-scale (e.g., Warsaw) using a microscopic model (TSF) – **(~30 seconds on standard machines)**.

➔ Time of inferencing neural network: **~0.8 ms** (time of training with GPU: ~10-15 minutes).

➔ And in the investigated case we don't even need large networks (3-4 layers with a few hundred neurons are sufficient).

➔ **We also achieved good results using LightGBM** (best avg error: ~1.72%, max error: ~10.83%, inference: ~0.4 ms).

Solution

Settings used in experiments:

➔ Inputs to NN: 21-element vector of signal settings (offsets on 21 crossroads, each offset is from the set {0, 1, 2, …, 119})

➔ Outputs of NN: approximated total waiting time of all vehicles in a given area (Stara Ochota district)

➔ Each NN was a feed forward fully connected NN with ReLU activation function (**tanh** turned out to be better)

➔ Configurations of hidden layers in neural networks: [100, 100, 100], [100, 200, 100], [200, 300, 200], [300, 400, 300], [100, 150, 200, 150, 100], [50, 100, 200, 300, 200, 100, 50]

➔ Values of a learning rate parameter: 0.1, 0.01, 0.001, 0.0001

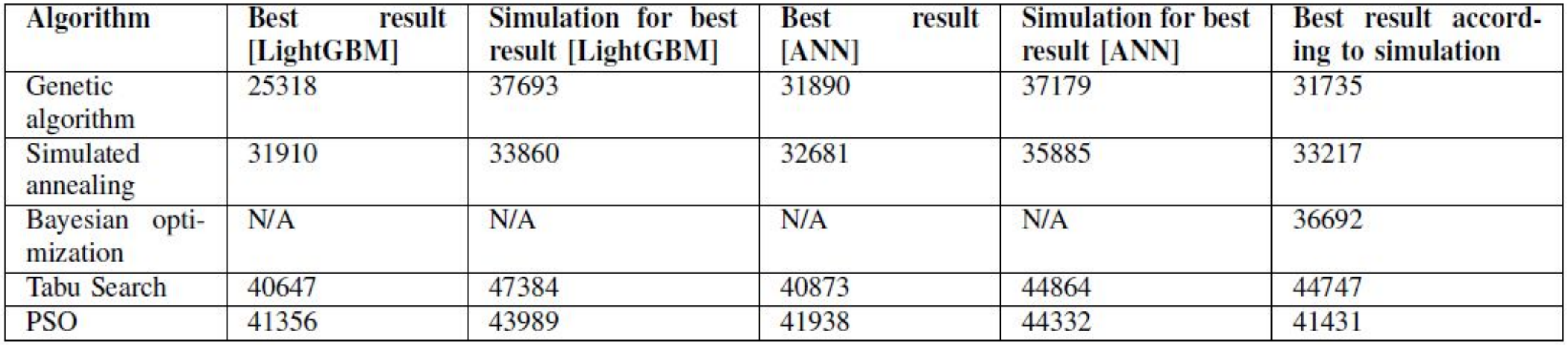
➔ Dropout probability (randomly removing some units to prevent overfitting): 0.05, 0.1, 0.15, 0.2

Application

We used both models as surrogate models (metamodels) evaluating traffic signal settings in traffic optimization algorithms.

6 algorithms tested:

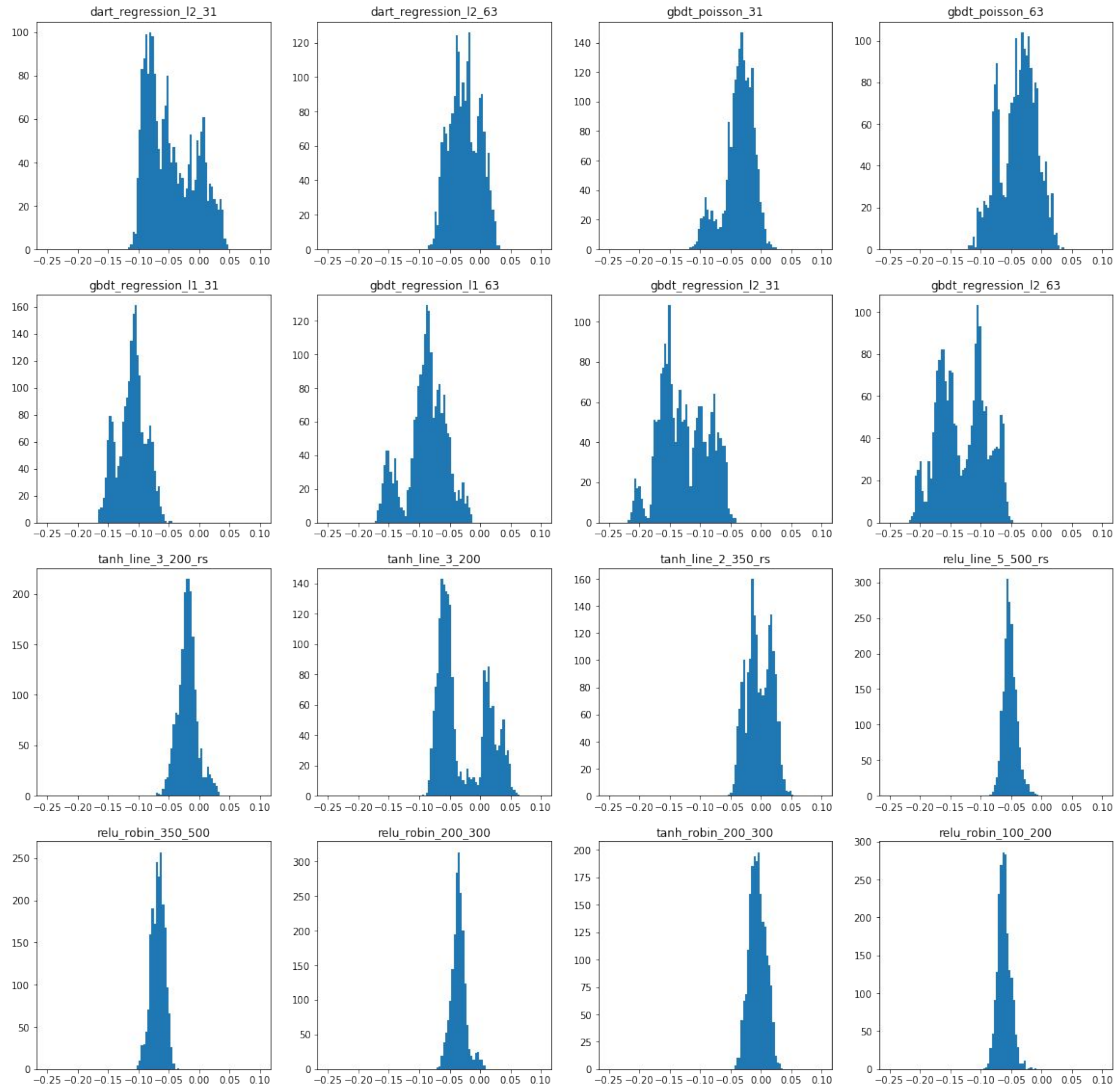
- Genetic algorithms
- Simulated annealing
- Particle swarm optimization
- Tabu search
- Bayesian optimization (without metamodels)
- Gradient optimization



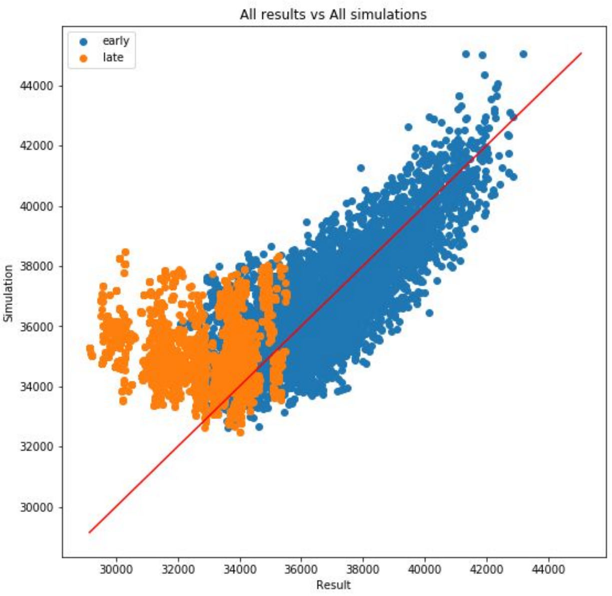
Algorithm	Best result [LightGBM]	Simulation for best result [LightGBM]	Best result [ANN]	Simulation for best result [ANN]	Best result according to simulation
Genetic algorithm	25318	37693	31890	37179	31735
Simulated annealing	31910	33860	32681	35885	33217
Bayesian optimization	N/A	N/A	N/A	N/A	36692
Tabu Search	40647	47384	40873	44864	44747
PSO	41356	43989	41938	44332	41431

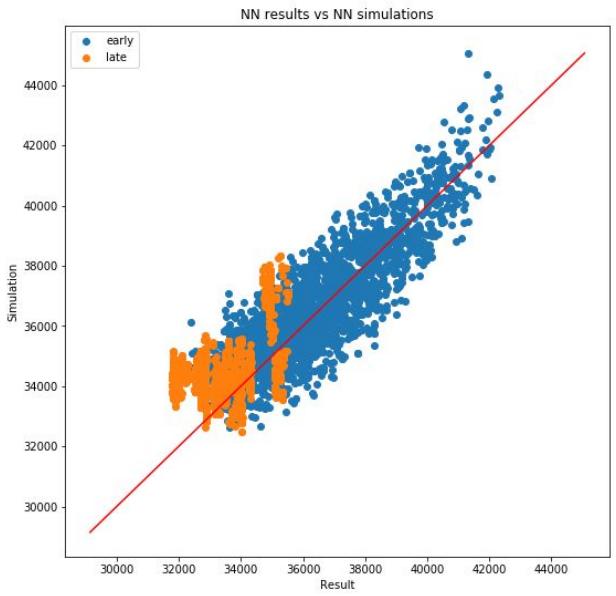
Comparison of best results found by optimization algorithms

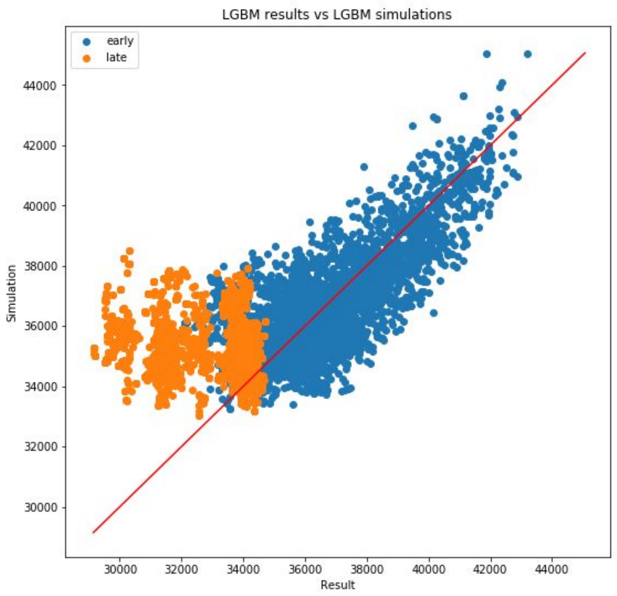
Distribution of error close to local optima



Results of a metamodel vs results of simulation for best settings from genetic algorithms (last 20 points from each run are marked orange)







Graph neural networks

Architecture I:

1. Neurons in the even numbered layers, starting with input layer as layer 0, should be localized at graph vertices (in our case - crossroads).

2. Neurons in the odd numbered layers should be localized at the graph edges (in our case - roads).

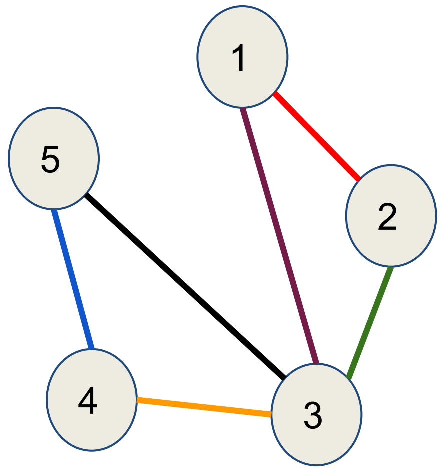
3. An exception should be the final layer with just one neuron.

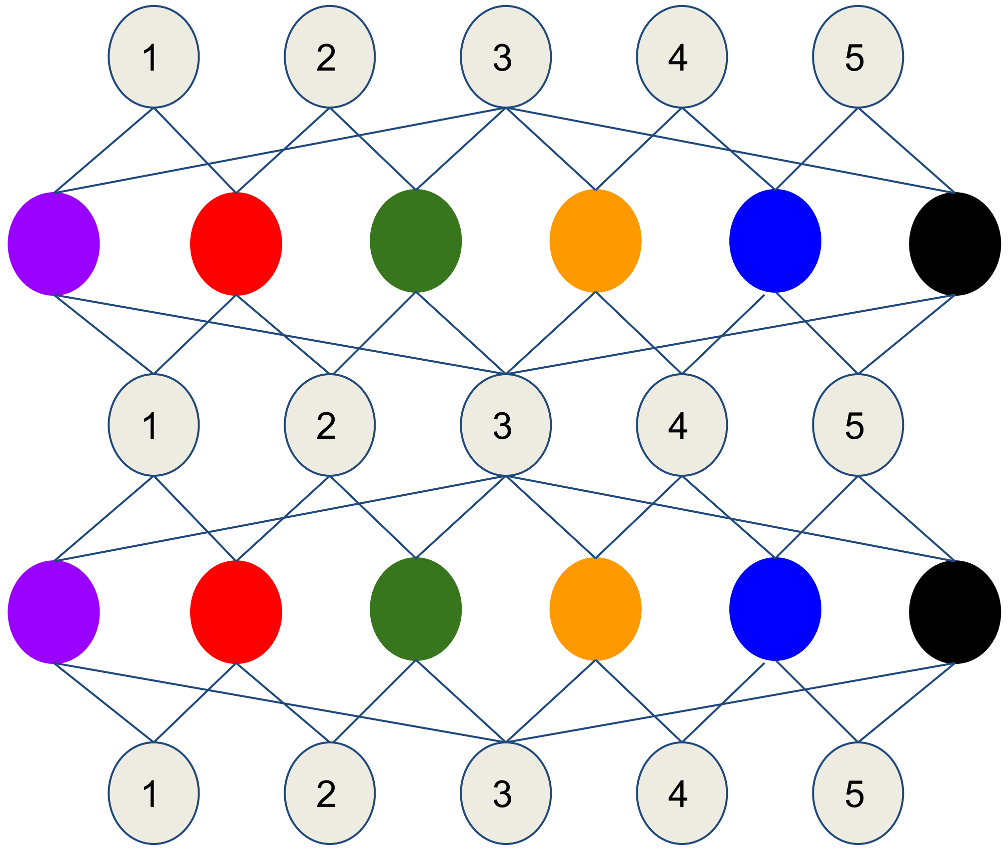
4. Connections from a vertex-localized layer to an edge-localized layer should only be present if a given vertex is an end of a given edge. There will be exactly two such connections for every edge neuron.

5. Connections from an edge-localized layer to a vertex-localized layer should only be present if the edge has the vertex as its end. The number of such connections will be equal to the number of particular vertex neighbors.

Graph neural networks

Architecture I:





Graph neural networks

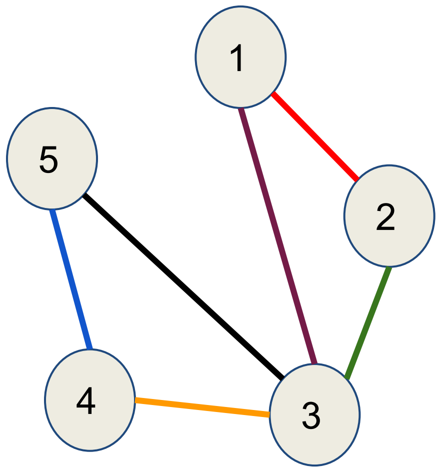
Architecture II:

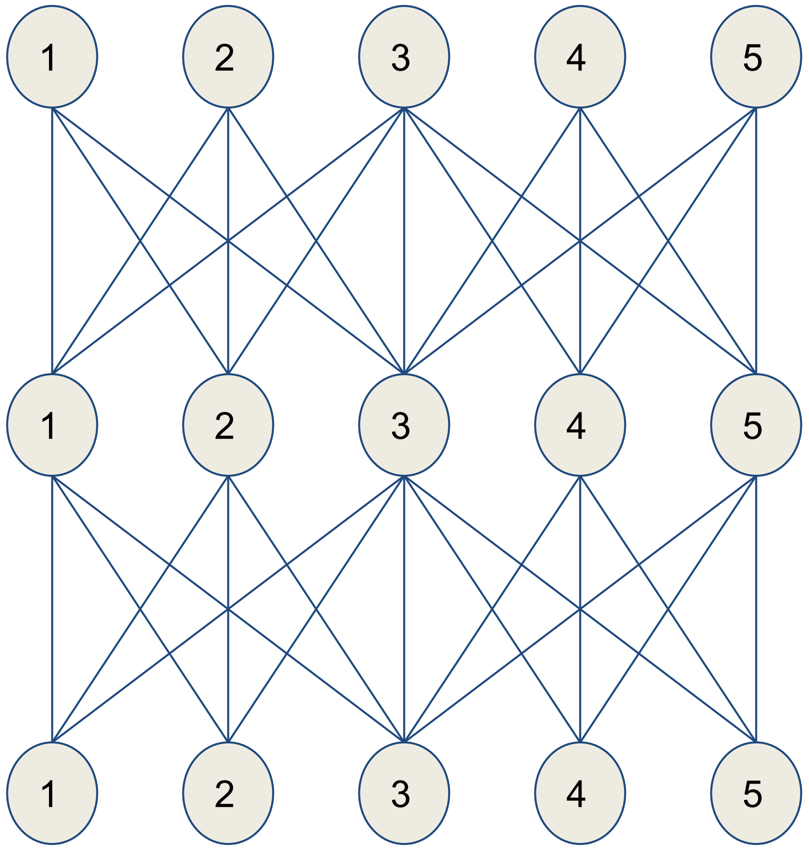
1. Neurons in all layers, with the exception of the output layer, should be localized at graph vertices.

2. Connections from a neuron in one layer to a neuron in the next one should only be present if the corresponding vertices are neighbors in the graph. The number of connections for the vertex node will be equal to the number of the vertex neighbors.

Graph neural networks

Architecture II:

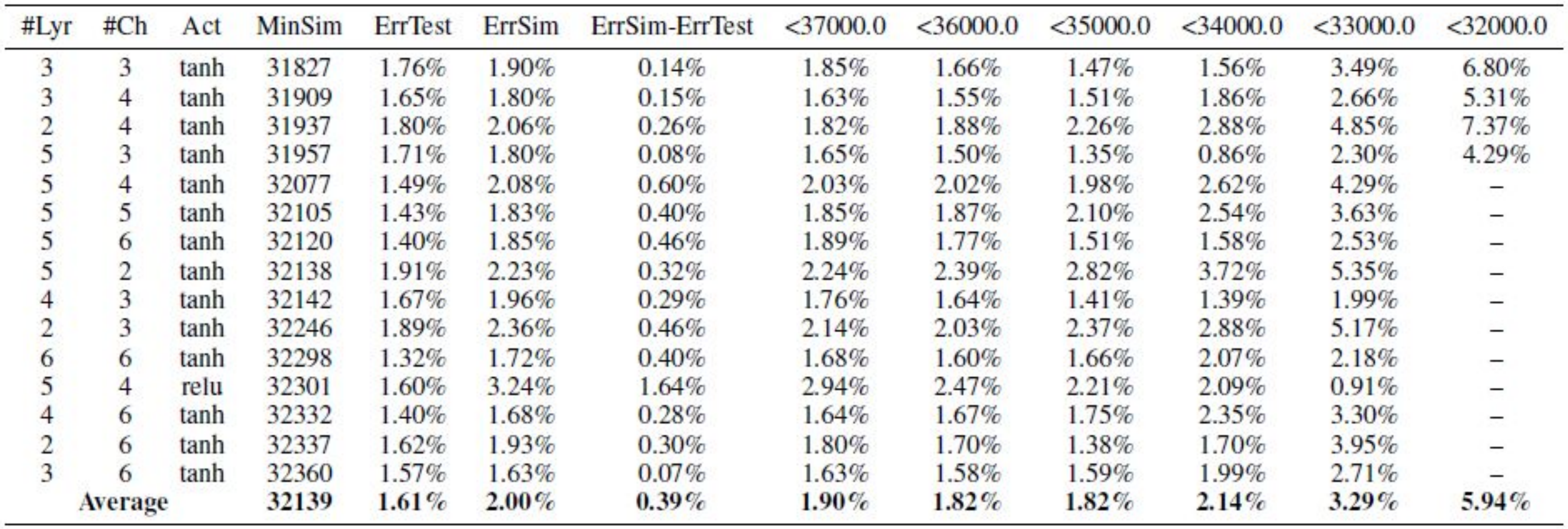




Graph neural networks

Results (Architecture I):

➔ reduced avg error of approximation on a test set (before: 1.62%, now: 1.32%),
➔ reduced max error of approximation on a test set (before: 10-11%, now: 6.18%),
➔ similar minima attained in the simulation (before: 31735, now: 31827),
➔ lower approximation error near minima



#Lyr	#Ch	Act	MinSim	ErrTest	ErrSim	ErrSim-ErrTest	<37000.0	<36000.0	<35000.0	<34000.0	<33000.0	<32000.0
3	3	tanh	31827	1.76%	1.90%	0.14%	1.85%	1.66%	1.47%	1.56%	3.49%	6.80%
3	4	tanh	31909	1.65%	1.80%	0.15%	1.63%	1.55%	1.51%	1.86%	2.66%	5.31%
2	4	tanh	31937	1.80%	2.06%	0.26%	1.82%	1.88%	2.26%	2.88%	4.85%	7.37%
5	3	tanh	31957	1.71%	1.80%	0.08%	1.65%	1.50%	1.35%	0.86%	2.30%	4.29%
5	4	tanh	32077	1.49%	2.08%	0.60%	2.03%	2.02%	1.98%	2.62%	4.29%	–
5	5	tanh	32105	1.43%	1.83%	0.40%	1.85%	1.87%	2.10%	2.54%	3.63%	–
5	6	tanh	32120	1.40%	1.85%	0.46%	1.89%	1.77%	1.51%	1.58%	2.53%	–
5	2	tanh	32138	1.91%	2.23%	0.32%	2.24%	2.39%	2.82%	3.72%	5.35%	–
4	3	tanh	32142	1.67%	1.96%	0.29%	1.76%	1.64%	1.41%	1.39%	1.99%	–
2	3	tanh	32246	1.89%	2.36%	0.46%	2.14%	2.03%	2.37%	2.88%	5.17%	–
6	6	tanh	32298	1.32%	1.72%	0.40%	1.68%	1.60%	1.66%	2.07%	2.18%	–
5	4	relu	32301	1.60%	3.24%	1.64%	2.94%	2.47%	2.21%	2.09%	0.91%	–
4	6	tanh	32332	1.40%	1.68%	0.28%	1.64%	1.67%	1.75%	2.35%	3.30%	–
2	6	tanh	32337	1.62%	1.93%	0.30%	1.80%	1.70%	1.38%	1.70%	3.95%	–
3	6	tanh	32360	1.57%	1.63%	0.07%	1.63%	1.58%	1.59%	1.99%	2.71%	–
Average			**32139**	**1.61%**	**2.00%**	**0.39%**	**1.90%**	**1.82%**	**1.82%**	**2.14%**	**3.29%**	**5.94%**

Best 15 rows in terms of the minimum simulator output value obtained by gradient descent. Columns #Lyr, #Ch and Act mean the number of layers, number of channels and activation function, respectively. Column MinSim includes the minimum output value of the simulator. Columns ErrTest, ErrSim and ErrSim-ErrTest contains average errors obtained on a test set, on a gradient descent trajectory and a difference between those average errors, respectively. The next 6 columns contain average errors on subsets of the trajectories obtained by thresholding on the simulator output values.

Graph neural networks

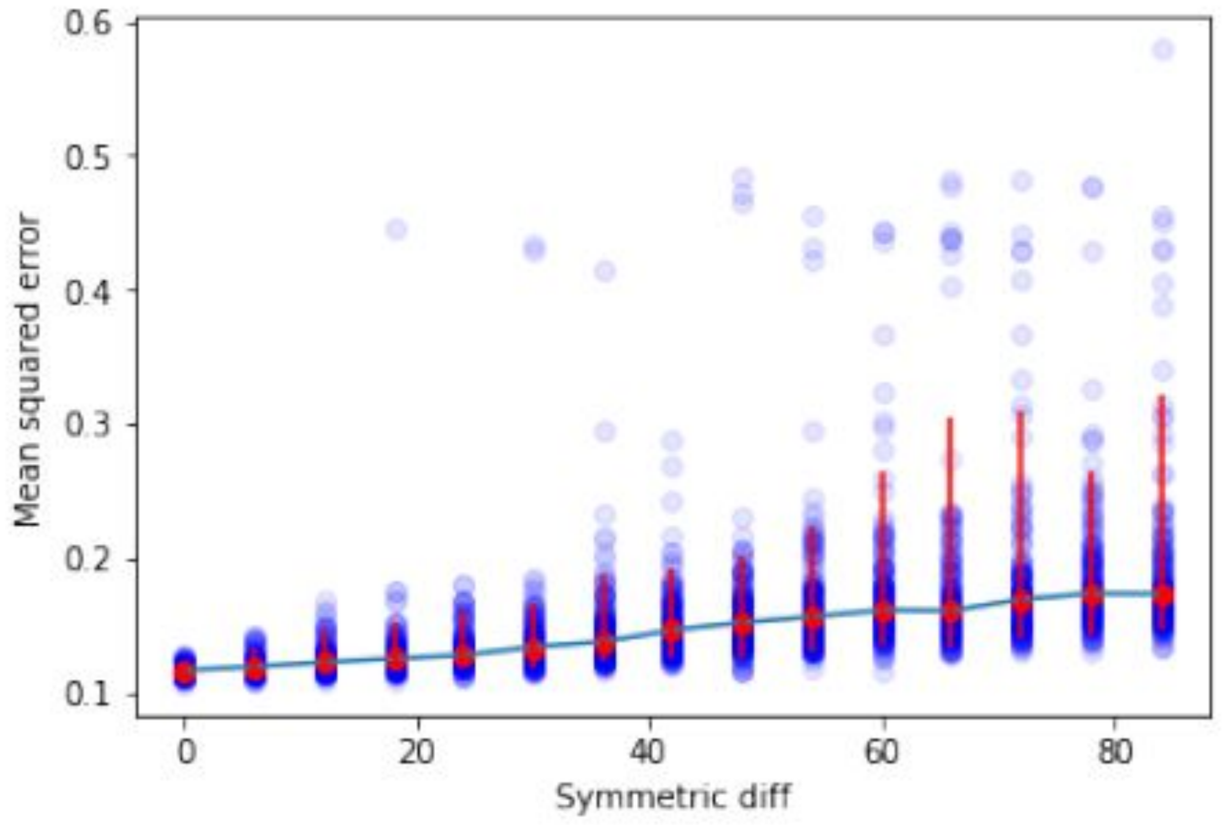
Problem:
It is not fully clear that including information about the topology of a road network brings any value. Perhaps, any similar graph, even not related to the problem at hand, could do equally well.

Sanity check:
We fixed the number of layers to 3 and the number of channels to 4 per layer (for architecture of type 1), and built our nets using random graphs with various degrees of similarity to the true problem graph (measure of similarity: symmetric difference between the sets of edges)

- Method 1: random edge insertions and deletions at the same time keeping the desired value of the symmetric difference.
- Method 2: random permutations of the vertex labels while keeping the connection graph structure exactly the same

Graph neural networks

Problem:
It is not fully clear that including information about the topology of a road network brings any value. Perhaps, any similar graph, even not related to the problem at hand, could do equally well.



Graph neural networks

Recent update:

➔ we introduced trigonometric normalization to make our models “learn” that signal offsets are cyclical

For traffic signal offset x in {0,1,2,...,119}:

$$x \rightarrow (\sin(2\pi x/120), \cos(2\pi x/120))$$

Graph neural networks

Why does it work (better than fully connected neural networks)?

Hypothesis:

- ➔ fewer parameters -> easier to train
- ➔ we keep only the most important connections between neurons (similarly to L1 regularization) -> easier to generalize

Traffic signal setting optimization

So, we can replace simulations with (graph) neural networks and use them as metamodels / surrogate models for evaluating fitness functions:

Optimization algorithms which we are testing:

- ➔ Genetic algorithms
  - ◆ Island models
  - ◆ Proximity-based crossover and mutation
- ➔ Simulated annealing
- ➔ Particle swarm optimization
- ➔ Gradient descent
- ➔ CMA-ES (Covariant Matrix Adaptation Evolutionary Strategy)
- ➔ Memetic algorithms
- ➔ Combinations of GA, SA, PSO, CMA-ES with gradient descent (or hill climbing)
- ➔ Memetic algorithms, GA and CMA-ES have given the best results so far

Applying BERT

- ➔ BERT is a transformer neural network using attention mechanism which has been successful in natural language processing

- ➔ We used a standard BERT model from the Hugging Face library

- ➔ It has been pretrained on Wikipedia and BookCorpus and can be used in number of downstream tasks like sentence and token classification, or question answering

- ➔ 2-step method:
  - ◆ classification on bucketized outputs
  - ◆ regression using fully connected neural network with dropout

- ➔ 1-step method: BERT is built into fully connected neural network which in each step of training generates embedding and performs a backpropagation with updating of BERT weights

Applying BERT

➔ We used Traffic Simulation Framework to generate a dataset set with 1.5mln evaluated signal settings (split into a training set, validation set and a test set with a proportion 80/10/10)

Applying BERT

- ➔ We used Traffic Simulation Framework to generate a dataset set with 1.5mln evaluated signal settings (split into a training set, validation set and a test set with a proportion 80/10/10)

- ➔ Settings were evaluated using TSF to compute the total times of waiting on red signals on Stara Ochota district in Warsaw (21 intersections)

Applying BERT

- ➔ We used Traffic Simulation Framework to generate a dataset set with 1.5mln evaluated signal settings (split into a training set, validation set and a test set with a proportion 80/10/10)

- ➔ Settings were evaluated using TSF to compute the total times of waiting on red signals on Stara Ochota district in Warsaw (21 intersections)

- ➔ Each signal setting consisted of 63 integer values, 3 values per intersection: durations of green signal phases in 2 directions (values from the set {20, 21, ... , 80}) and an offset (values from the set {0, 1, 2, ... ,max})

Applying BERT

- ➔ We used Traffic Simulation Framework to generate a dataset set with 1.5mln evaluated signal settings (split into a training set, validation set and a test set with a proportion 80/10/10)

- ➔ Settings were evaluated using TSF to compute the total times of waiting on red signals on Stara Ochota district in Warsaw (21 intersections)

- ➔ Each signal setting consisted of 63 integer values, 3 values per intersection: durations of green signal phases in 2 directions (values from the set {20, 21, ... , 80}) and an offset (values from the set {0, 1, 2, ... ,max})

- ➔ We trained and compared BERT, LightGBM, fully connected neural networks (FCNN), Graph Convolutional Networks (GCN) and Graph Neural Networks (GNN)

Applying BERT

- ➔ We used Traffic Simulation Framework to generate a dataset set with 1.5mln evaluated signal settings (split into a training set, validation set and a test set with a proportion 80/10/10)
- ➔ Settings were evaluated using TSF to compute the total times of waiting on red signals on Stara Ochota district in Warsaw (21 intersections)
- ➔ Each signal setting consisted of 63 integer values, 3 values per intersection: durations of green signal phases in 2 directions (values from the set {20, 21, ... , 80}) and an offset (values from the set {0, 1, 2, ... ,max})
- ➔ We trained and compared BERT, LightGBM, fully connected neural networks (FCNN), Graph Convolutional Networks (GCN) and Graph Neural Networks (GNN)
- ➔ The results of training were evaluated on a test set and compared using 3 metrics: MAPE, MAXPE (maximum percentage error), MAXPE99 (maximum percentage error among best 99% results - 99-th percentile).

Applying BERT

Model	**MAPE**	**MAXPE**	**MAXPE99**
LightGBM	0.034	0.271	0.117
FCNN (63, 256, 128, 64, 48, 32, 16, 8, 1)	0.037	0.327	0.132
GCN (2 GCN layers + FCNN (21, 128, 48, 32))	0.078	0.442	0.251
GNN (1 GNN layer + FCNN (63, 128, 64, 32, 1))	0.069	0.416	0.229
GNN (1 GNN layer + FCNN (63, 256, 128, 64, 32, 8 , 1))	0.069	0.423	0.229
GNN (3 GNN layers + FCNN (63, 256, 128, 64, 32, 8 , 1))	0.069	0.423	0.228
BERT (1 step)	**0.019**	**0.178**	**0.066**
BERT (2 step)	0.02	0.223	0.108

Applying BERT

The paper “Predicting times of waiting on red signals using BERT” which I prepared together with **Witold Szejgis** and **Anna Warno** has been accepted for the **NeurIPS 2020 workshop "Machine Learning for Autonomous Driving"**: https://ml4ad.github.io.

Why pretrained BERT gives good results in this case?

- Random initialization? No.
- Order of crossroads in the input vector? No.
- ?

Traffic signal setting optimization

Summary:

- Done:
  - Trained **graph neural networks**, **LightGBM,** feed forward NN, random forests, SVM for the task of approximating outcomes of traffic simulation
  - **Trained BERT** (and graph neural networks, LightGBM, GCNN) for a dataset with varying durations of green signal phases
  - Several metaheuristics tested with **graph neural networks** on the datasets for Ochota, Centrum, Mokotów

Traffic signal setting optimization

Summary:

- ➔ Done:
  - ◆ Trained **graph neural networks**, **LightGBM,** feed forward NN, random forests, SVM for the task of approximating outcomes of traffic simulation
  - ◆ **Trained BERT** (and graph neural networks, LightGBM, GCNN) for a dataset with varying durations of green signal phases
  - ◆ Several metaheuristics tested with **graph neural networks** on the datasets for Ochota, Centrum, Mokotów
- ➔ In progress:
  - ◆ **Several metaheuristics are tested with LightGBM on the datasets for Ochota, Centrum, Mokotów**
  - ◆ Writing thesis (should I include more experiments for BERT?)
  - ◆ Writing chapter for the “AI in Intelligent Transportation Systems” book

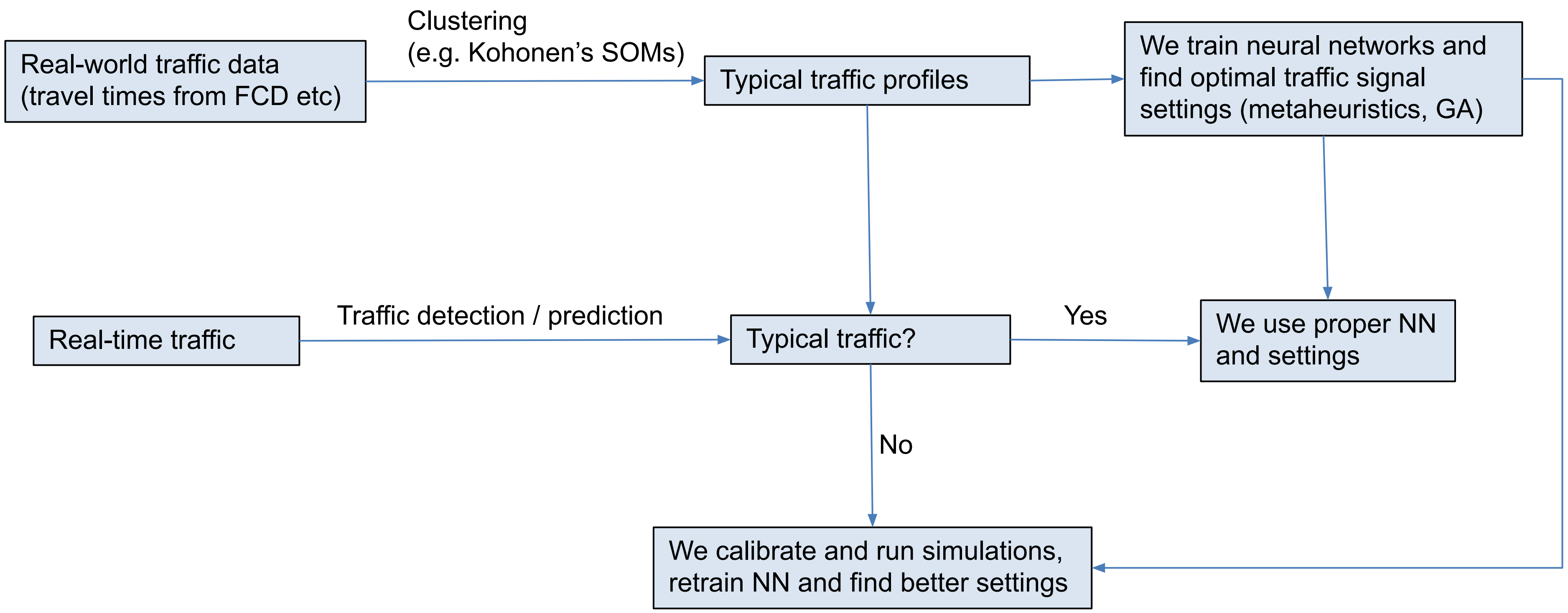
Traffic signal setting optimization

Summary:

- ➔ Done:
  - ◆ Trained **graph neural networks**, **LightGBM,** feed forward NN, random forests, SVM for the task of approximating outcomes of traffic simulation
  - ◆ **Trained BERT** (and graph neural networks, LightGBM, GCNN) for a dataset with varying durations of green signal phases
  - ◆ Several metaheuristics tested with **graph neural networks** on the datasets for Ochota, Centrum, Mokotów
- ➔ In progress:
  - ◆ **Several metaheuristics are tested with LightGBM on the datasets for Ochota, Centrum, Mokotów**
  - ◆ Writing thesis (should I include more experiments for BERT?)
  - ◆ Writing chapter for the “AI in Intelligent Transportation Systems” book
- ➔ Future work:
  - ◆ Transfer learning (to other traffic conditions or to other road networks)
  - ◆ Active learning (adding heuristically optimal points to the dataset)
  - ◆ Clustering of traffic states
  - ◆ Scaling the solution to larger areas (new datasets will be generated using a Sobol sequence)
  - ◆ Testing ensembles

Applications

AI-based (and simulation-based) real-time traffic management


Real-world traffic data
(travel times from FCD etc)
Clustering
(e.g. Kohonen’s SOMs)
Typical traffic profiles
We train neural networks and find optimal traffic signal settings (metaheuristics, GA)
Real-time traffic
Traffic detection / prediction
Typical traffic?
Yes
We use proper NN and settings
No
We calibrate and run simulations, retrain NN and find better settings

Applications

We can apply AI and traffic simulations to :

- optimize locations and capacities of parkings (even ~25-30% of traffic in city centers is generated by drivers looking for parking place)
- optimize locations of charging stations for electric vehicles
- optimize road network structure
- optimize algorithms for connected and autonomous vehicles
- optimize tolling strategies
- optimize variable speed limits (project “INZNAK”)
- …

Those methods may be especially efficient in a large scale, in case of atypical traffic and in the era of connected and autonomous vehicles.

Reinforcement learning

Idea:

- Environment - traffic simulation.
- State - state of the traffic (e.g., vector with numbers of cars and average speeds for some road segments).
- Action - modification of traffic signal settings.
- Reward - quality of traffic in a given time period, e.g., the total time of waiting, the total travelled distance, the amount of consumed energy.

Reinforcement learning

Idea:

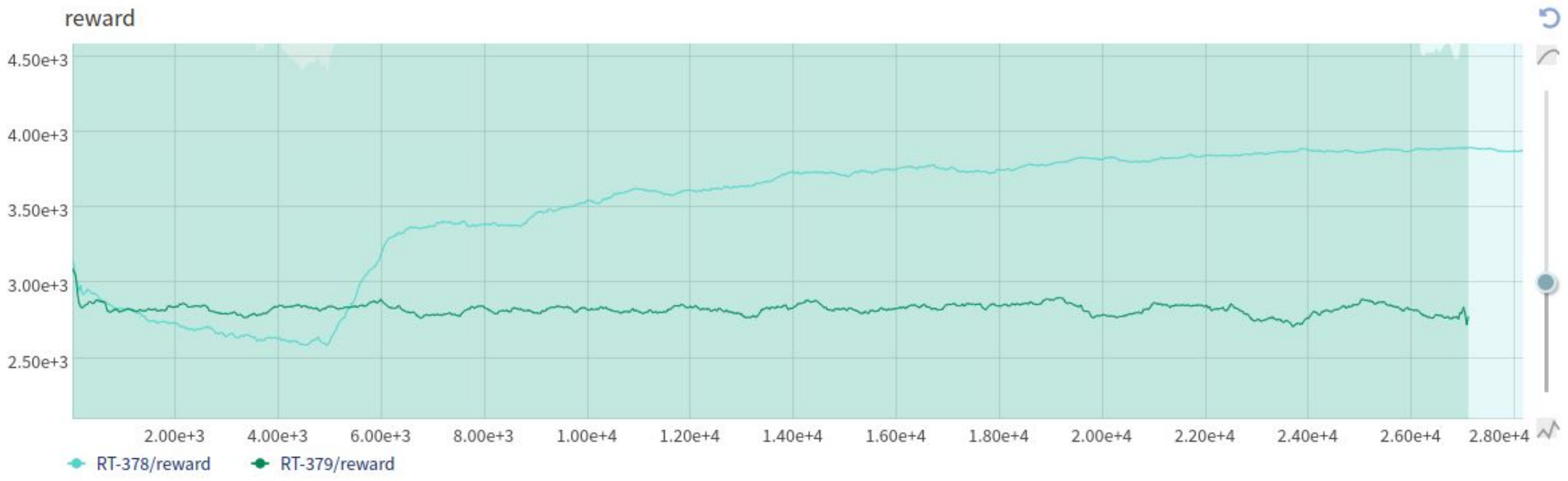
- Environment - traffic simulation.
- State - state of the traffic (e.g., vector with numbers of cars and average speeds for some road segments).
- Action - modification of traffic signal settings.
- Reward - quality of traffic in a given time period, e.g., the total time of waiting, the total travelled distance, the amount of consumed energy.

Experiments:

- **Environment** - we implemented a simple traffic simulator based on Na-Sch (deterministic).
- **State** - vector of average speeds and number of cars (per road segment) for one cycle of traffic lights.
- **Action** - 4-element vector of offsets (offset indicates when the green light starts).
- **Reward** - the total distance covered by all cars in 1 step.
- **Algorithm:** DQN.

Reinforcement learning

Recent results

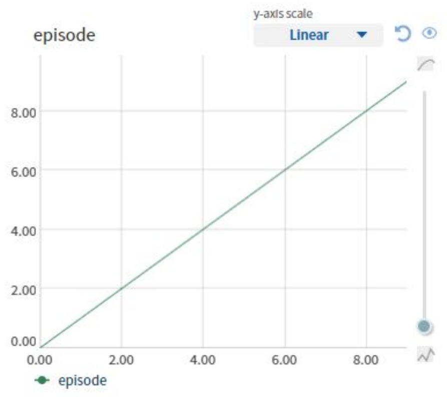


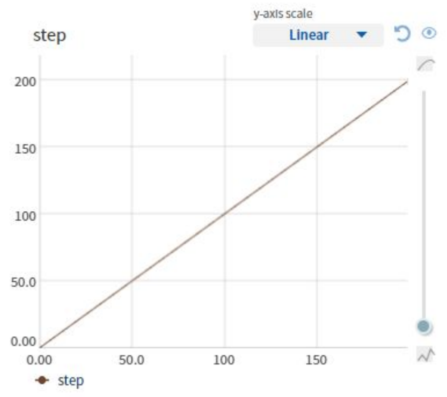
X axis is a step
light green is RL approach
dark green is baseline (agent selects actions randomly)

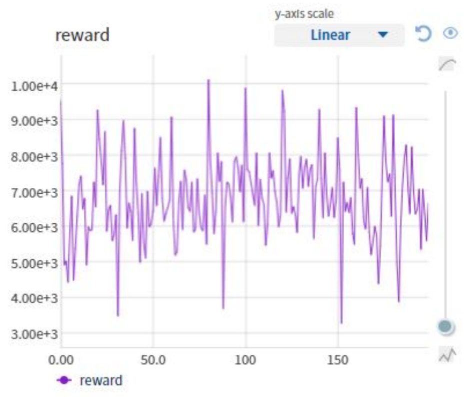
Reinforcement learning

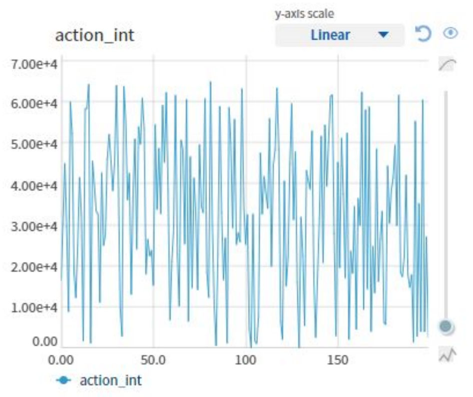
Recent results

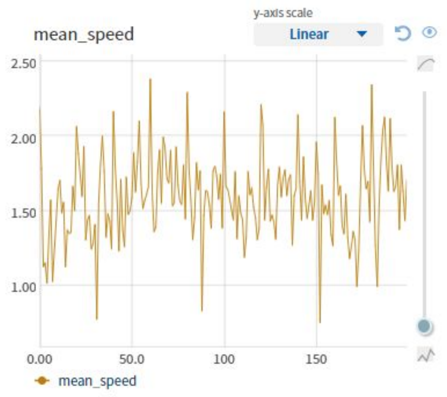
- ➔ Integration with Flow https://flow-project.github.io
- ➔ Integration with OpenAI Gym
- ➔ Integration with libraries RLlib, stable-baselines
- ➔ Testing PPO, TRPO, ARS
- ➔ Integration with Neptune.ai

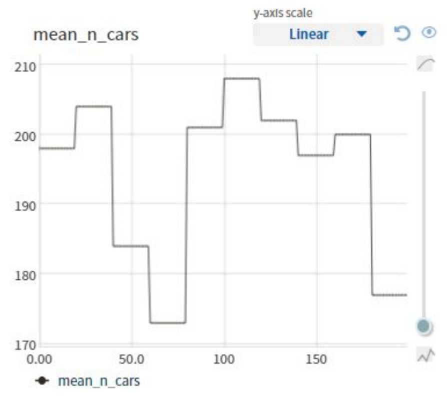


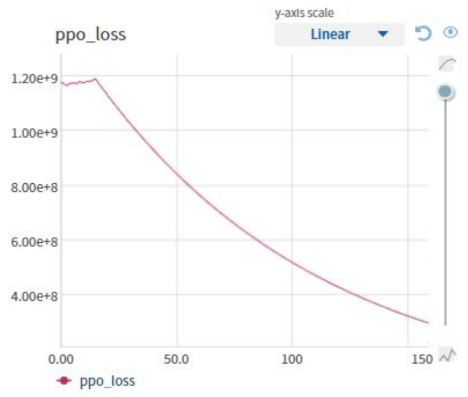










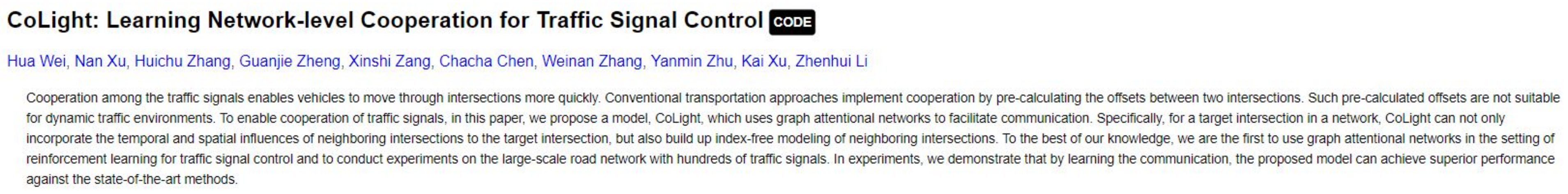


Reinforcement learning

Recent results

➔ Multiagent RL

➔ “CoLight: Learning Network-level Cooperation for Traffic Signal Control” (graph attention networks)



CoLight: Learning Network-level Cooperation for Traffic Signal Control CODE

Hua Wei, Nan Xu, Huichu Zhang, Guanjie Zheng, Xinshi Zang, Chacha Chen, Weinan Zhang, Yanmin Zhu, Kai Xu, Zhenhui Li

Cooperation among the traffic signals enables vehicles to move through intersections more quickly. Conventional transportation approaches implement cooperation by pre-calculating the offsets between two intersections. Such pre-calculated offsets are not suitable for dynamic traffic environments. To enable cooperation of traffic signals, in this paper, we propose a model, CoLight, which uses graph attentional networks to facilitate communication. Specifically, for a target intersection in a network, CoLight can not only incorporate the temporal and spatial influences of neighboring intersections to the target intersection, but also build up index-free modeling of neighboring intersections. To the best of our knowledge, we are the first to use graph attentional networks in the setting of reinforcement learning for traffic signal control and to conduct experiments on the large-scale road network with hundreds of traffic signals. In experiments, we demonstrate that by learning the communication, the proposed model can achieve superior performance against the state-of-the-art methods.

Optimization of cancer treatment

➔ Cooperation with dr hab. Monika Piotrowska (MIMUW), Rafał Banaś (MIMUW), Wojciech Ozimek (UJ/Ardigen), Simon Angus (Monash University)

Optimization of cancer treatment

- ➔ Cooperation with dr hab. Monika Piotrowska (MIMUW), Rafał Banaś (MIMUW), Wojciech Ozimek (UJ/Ardigen), Simon Angus (Monash University)
- ➔ We can simulate cancer growth using cellular automata (stochastic model)
- ➔ We can apply different radiotherapy strategies (times and doses of radiotherapy) ...
- ➔ … and compute the number of tumor cells
- ➔ We can try to find good strategies using genetic algorithms (or other metaheuristics)

Optimization of cancer treatment

- Cooperation with dr hab. Monika Piotrowska (MIMUW), Rafał Banaś (MIMUW), Wojciech Ozimek (UJ/Ardigen), Simon Angus (Monash University)
- We can simulate cancer growth using cellular automata (stochastic model)
- We can apply different radiotherapy strategies (times and doses of radiotherapy) ...
- … and compute the number of tumor cells
- We can try to find good strategies using genetic algorithms (or other metaheuristics)
- Accelerating computations:
  - Initially (Matlab): ~100 minutes per simulation (of 10 days of treatment)
  - Our first try (C++): ~1 minute per simulation
  - Second try (CUDA): a few second per simulation
  - Neural networks / LightGBM: < 0.1 ms (Temporal Fusion Transformer gives the best results)

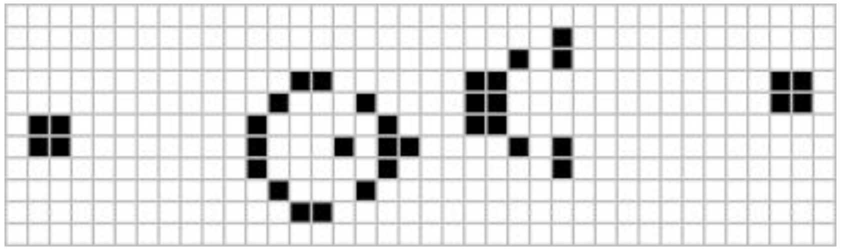
Optimization of cancer treatment

- Cooperation with dr hab. Monika Piotrowska (MIMUW), Rafał Banaś (MIMUW), Wojciech Ozimek (UJ/Ardigen), Simon Angus (Monash University)
- We can simulate cancer growth using cellular automata (stochastic model)
- We can apply different radiotherapy strategies (times and doses of radiotherapy) ...
- … and compute the number of tumor cells
- We can try to find good strategies using genetic algorithms (or other metaheuristics)
- Accelerating computations:
  - Initially (Matlab): ~100 minutes per simulation (of 10 days of treatment)
  - Our first try (C++): ~1 minute per simulation
  - Second try (CUDA): a few second per simulation
  - Neural networks / LightGBM: < 0.1 ms (Temporal Fusion Transformer gives the best results)
- We can use reinforcement learning to adapt times and values of doses to the current state

Approximating outcomes of cellular automata evolution

Some CA are Turing-complete (Game of Life, Rule 110)

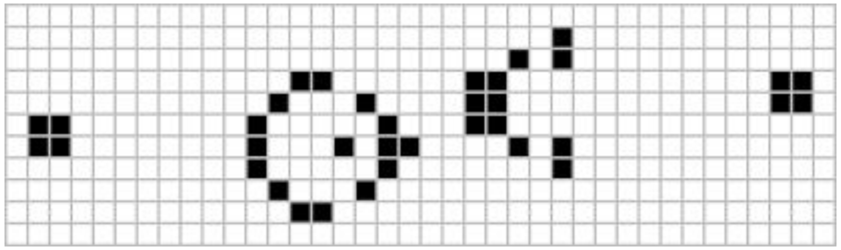
Can we build a universal tool for finding approximate solutions of (some) computational problems (e.g., NP-hard problems)?



Approximating outcomes of cellular automata evolution

Some CA are Turing-complete (Game of Life, Rule 110)

Can we build a universal tool for finding approximate solutions of (some) computational problems (e.g., NP-hard problems)?



"It's Hard for Neural Networks To Learn the Game of Life"

https://bdtechtalks.com/2020/09/16/deep-learning-game-of-life

https://arxiv.org/abs/2009.01398

Thank you for your attention!

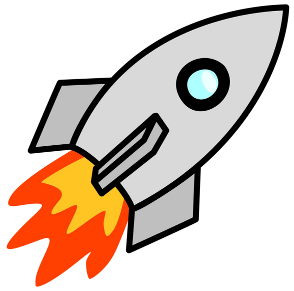
Questions?

E-mail: p.gora@mimuw.edu.pl
tensorcell.research@gmail.com

www: http://www.mimuw.edu.pl/~pawelg
http://www.tensorcell.com

“Logic can get you from A to B, imagination will take you everywhere” A. Einstein

*“The sky is **NOT** the limit”*



Future work

- ➔ New training sets: Centrum, Mokotów
- ➔ Hierarchical approach
- ➔ Active learning
- ➔ Ensemble learning
- ➔ Transfer learning
- ➔ AutoML (finding optimal NN structures and hyperparameters using AI)

Acknowledgement

TensorCell (current or past) members / contributors:

- Łukasz Skowronek
- Marcin Możejko
- Arkadiusz Klemenko
- Maciej Brzeski
- Kamil Kaczmarek
- Anna Kosiorek
- Dawid Kopczyk
- Katarzyna Karnas
- Łukasz Mądry
- Karol Kurach
- Marek Bardoński
- Mateusz Susik
- Magdalena Kukawska
- Maciej Zwoliński
- Mariusz Patyk
- Hubert Dryja
- Przemysław Przybyszewski
- Adrian Kochański
- Piotr Golach
- ...

TensorCell

- ➔ An independent research group founded in 2016
- ➔ We work on optimizing complex processes using AI
- ➔ 100% remote work
- ➔ > 20 researchers (mostly from Warsaw, 6 from Kraków, 1 from Poznań)
- ➔ Cooperation with researchers from Google, Delft University (in the past also: Skolkovo Institute of Science and Technology, Technical University of Madrid, Edinburgh Napier University)
- ➔ Azure resources from Microsoft (“AI for Earth” grant)
- ➔ Papers published on NeurIPS workshops, MT-ITS, TFML etc

Optimizing complex processes using AI

Complex process - (evolving in time) system composed of many interacting elements.

Examples: vehicular traffic, society, ant colony, organism (e.g., evolving cells), human brain, cellular automata, stock market, …

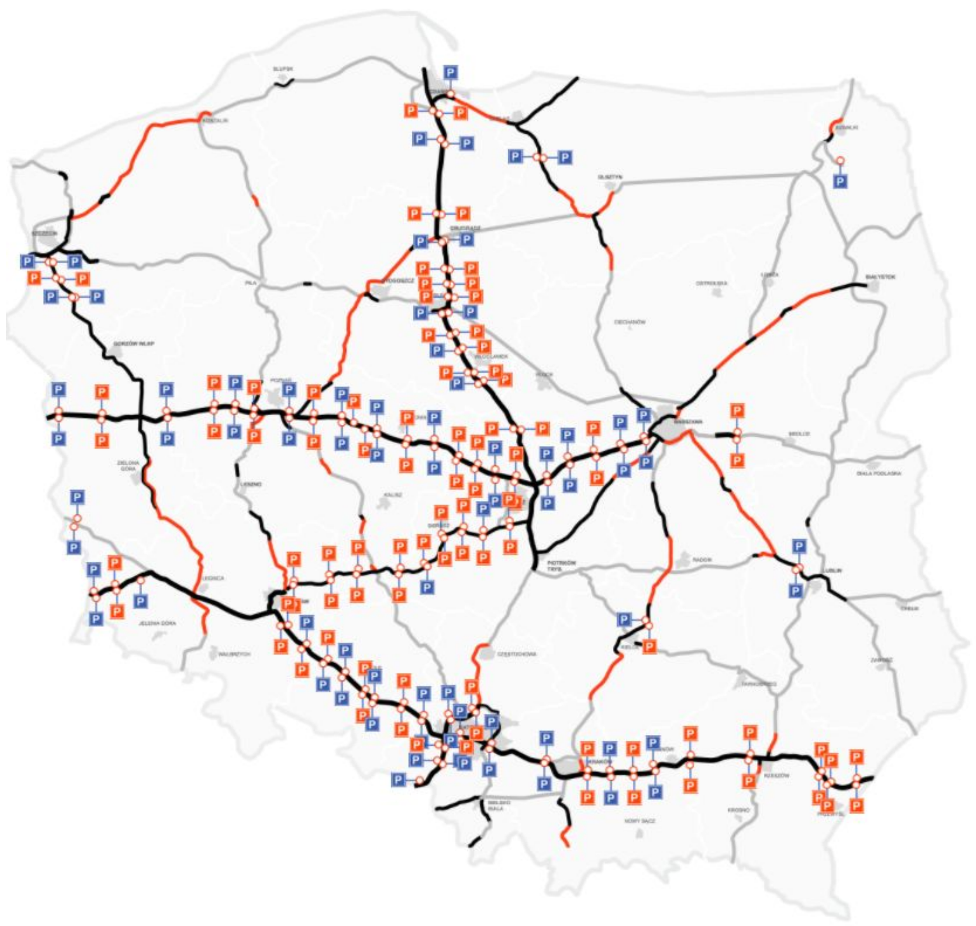
TensorCell

Typical properties of complex systems / processes:

- Emergent properties (traffic jam, war, financial bubble / crisis)
- Nonlinearity
- Sensitive dependence on initial conditions
- Openness (influence of the environment)
- Adaptability
- Hierarchy
- Computational irreducibility
- “Order in chaos” - between predictable and random behaviour

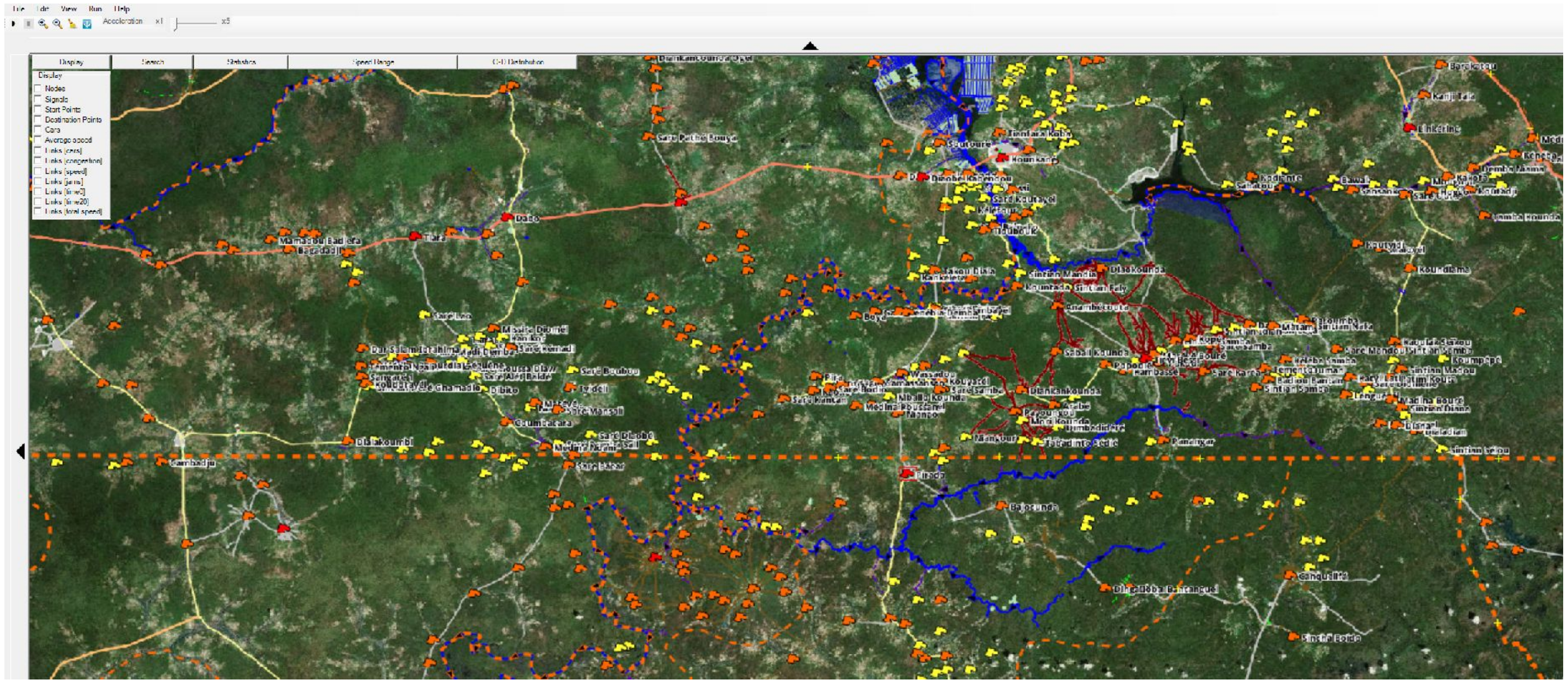
Fascinating universality and ubiquitousness!

Applications



Rest areas in Poland
Source: GDDKiA

Applications



TSF applied to find optimal locations of a bridge close to the Senegal - Guinea border