Transport network design for vehicle routing: results on path addition and capacity reduction

Matteo Bettini
Sidney Sussex College

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Department of Computer Science and Technology
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: mb2389@cam.ac.uk

June, 2021

Declaration

I, Matteo Bettini of Sidney Sussex College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. Appendix A has been excluded from the word count.

Word count: 14731

Signed: Matteo Bettini

Date: June 4, 2021

Acknowledgements

Throughout the research process for this dissertation I have received a great deal of support and assistance.

I would first like to thank my supervisor, Dr. Amanda Prorok, whose expertise was invaluable in the formulating of the research topic. She allowed this paper to be my own work, but steered me in the right the direction whenever she thought I needed it.

I would also like to thank my co-supervisor, Matthew Malencia, for all the support and guidance. I particularly enjoyed our frequent conversations which often lasted for hours, discussing interesting research directions and exciting new problems.

Another thanks goes to all the members of ProrokLab, which have been of great inspiration during this year.

Finally, I must express my very profound gratitude to my parents and to my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Transport network design for vehicle routing: results on path addition and capacity reduction

Abstract

Autonomous Vehicles (AVs) are becoming increasingly popular thanks to advancements in Artificial Intelligence algorithms that enable self-driving. This technological shift will lead to a drastic change in our current mobility paradigms. Historic trends suggest that this change will heavily impact the transportation infrastructure. While a lot of works focus on control and routing of AVs, little attention is being paid to the transportation network. In this thesis, we investigate the problem of optimising the transportation network for AVs both from a theoretical and from a practical perspective. In the first part, we investigate the properties of adding paths to a network and prove that the intuitive statement *adding a path to a transport network always grants greater or equal benefit to users than adding it to a bigger network* is false. In other words, we prove that path additions to transport networks, where AVs are routed, are not supermodular in travel time, extending the seminal result of Braess' paradox. We provide counterexamples to support our proofs. We further provide some formulations where, instead, we are able to prove the supermodularity of path additions. In the second part, we formulate two network design problems for self-interested AVs. We present the problem of optimising transport networks via path additions and a novel problem design where self-interested users are guided towards optimal paths through the reduction of road capacities. This formulation, unlike previous network design research, allows users to see non-optimal roads as more costly by bridging a road pricing policy with environment shaping. To solve the capacity reduction problem, we implement a genetic algorithm as well as a reinforcement learning task using a designer agent trained with proximal policy optimisation and a graph neural network architecture. We simulate our solutions in the microscopic traffic simulator SUMO and in a custom built macroscopic traffic simulator. Through capacity reductions, we achieve significant total travel time improvements on six real-world transport networks: Anaheim (USA), Barcelona (Spain), Chicago (USA), Eastern Massachusetts (USA), Sioux Falls (USA), and Winnipeg (Canada). For instance, we improve the Chicago network by up to 7%, saving more than 487 hours of total travel time per traffic hour. This is done strictly through virtual capacity reductions, without the need for physical or infrastructural modifications to the network. This makes our solutions ready for immediate deployment on any transport network in the world.

Word count

Total word count: 14731

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous Vehicles (AVs) are attracting increasing attention thanks to their revolutionary potential to improve current mobility paradigms. Recent progress in the design of electric engines and of artificial intelligence algorithms deployed on board of AVs is contributing to accelerate this mobility shift. As these vehicles will require no human input, we are preparing to a transition in which human control will make room for cooperation among AVs.

As we can observe from historic trends, every change to our mobility systems has been followed by a change to the transportation network infrastructure. In this work, we want to anticipate this trend by investigating the impact that the transportation network infrastructure has on AVs' routing. In particular, drawing from the insights of the famous Braess' paradox [1], we tackle the problem of designing transportation networks for AVs from a range of different theoretical and practical perspectives.

Unlike past network design research, which focuses on adding physical infrastructure to transport networks under a construction budget, we focus on pushing the single AV's interest towards the system optimum just by shaping the current network and by removing infrastructure.

In the first part of this project, we tackle the problem of AVs routing from a general theoretical perspective. We model AVs as flow traversing a transport network represented as a graph. We prove that the intuitive statement *adding a path to a transport network always grants greater or equal benefit to users than adding it to a bigger network* is false. In other words, we prove that path additions to transport networks where AVs are routed, are not supermodular in travel time. This is valid both when vehicles are routed according to their own interest and when they are routed according to the system's optimum. This result extends Braess' paradox, which proved the non-monotonicity of the same problem for self-interested users. As in Braess' work [1], we provide counterexamples to support our proofs. This result is proven both for congestion-aware AVs and for congestion-agnostic AVs (traditional min-cost flow problems). We furtherly provide and prove some scenarios where, instead, the property of supermodularity holds. The results of this theoretical investigation are something that every network designer should be aware of, as they provide important proofs that an intuitive property of path additions is indeed false.

In the second part of this project, we formalise two network design tasks as bilevel

optimisation problems. In these problems, the network designer can modify the transport network topology and observe how AVs redistribute in the new network. For the first task, we start from a spanning tree transport network and try to optimally add paths to it. For the second task, we start with a network topology (e.g. a city), where we believe that Braess' paradox occurs, and we seek to reduce the edge capacities in order to make the selfish AVs behave optimally (minimise the total system travel time). Our capacity reduction is similar to a road pricing scheme, in the sense that we show some roads to users as more costly than they really are, but costs are not fictitious, as they are always mapped to edge capacities, which are real topology parameters. To solve the first problem, we implement a greedy algorithm, while, for the second, we implement a Genetic Algorithm (GA) as well as a Reinforcement Learning (RL) task using a designer agent trained with Proximal Policy Optimisation (PPO) and a Graph Neural Network (GNN) architecture. We simulate our solutions in the microscopic traffic simulator SUMO [2] and in a custom built macroscopic simulator. To solve the routing problem, we reproduce self-aware routing [3] and also implement the Frank-Wolfe traffic assignment algorithm [4]. The macroscopic simulator and the traffic assignment solver have been published as standalone products and are already being used by other researchers. In this simulator, we test our capacity reduction algorithms on six real-world transport networks: Anaheim (USA), Barcelona (Spain), Chicago (USA), Eastern Massachusetts (USA), Sioux Falls (USA), and Winnipeg (Canada). Our results show significant improvements in total travel time on all networks, with hundreds of travel hours saved while maintaining fairness to the users at the routing level.

The network design formulation we provide is innovative and could help traffic control organisations, such as governments, optimise AVs' traffic throughput. In a scenario where different AVs' producers deploy self-interested vehicles, our capacity reduction network design framework could be implemented by the central organisation to push AVs to behave optimally, while maintaining fairness to the single users. Our solutions are ready for immediate deployment on any transport network in the world. Furthermore, all of our insights can be applied to any network flow model. One example is crowd control [5], which has become increasingly relevant after the COVID-19 pandemic.

The dissertation is organised as follows: Chapter 2 introduces some background concepts, Chapter 3 analyses the relevant related work, Chapter 4 discusses the supermodularity of path additions, Chapter 5 introduces our network design tasks, Chapter 6 contains the evaluation of the proposed solutions, and, lastly, Chapter 7 concludes this dissertation.

Chapter 2

Background

In this chapter, we illustrate the main background concepts underlying the development of this project. The foundation of this work stems from the field of traffic research and, in particular, traffic assignment. We draw from seminal contributions such as the Wardrop equilibrium [6] and the Braess' paradox [1] to develop our theoretical results. We apply these concepts to real-world problems formulated as optimisation tasks. To tackle such problems, we employ traditional techniques from the field of operations research and evolutionary computation as well as innovative approaches through the use of graph neural networks and reinforcement learning.

2.1 Transportation networks

The environment in which vehicles are routed can be described as a transportation network (also known as transport network). A transport network is a realisation of a spatial network, characterising a structure which permits agent movement. Some examples are: road networks, railway networks and air routes. In such networks the congestion level is defined as the user load on a specific segment. Transport networks can be formally modeled as graphs whose nodes represent intersections and edges represent paths connecting them (e.g., roads). Different properties can be assigned to nodes and edges. In this work, we associate to each edge a cost function representing the travel time required to traverse that edge, which is *dependent on its current congestion*. We also consider edge capacities, which represent the maximum congestion level suitable for a specific edge.

The main scenarios described in this project consider road networks where autonomous cars are routed. However, the flow problem formulation that we use does not bind our results to this agent type and transport infrastructure. In fact, our insights could be applied to any transportation network structuring problem. One example is describing and controlling crowd movements as flow distributing in pedestrian transport network [5], a problem that has seen increasing attention during the COVID-19 pandemic.

2.2 Traffic and congestion

2.2.1 System optimal and user equilibrium routing

Traffic can be modeled as flow in a transport network, represented as a graph. Under this formulation, user trips are defined as origin-destination (OD) pairs in the network with an associated demand that represents the amount of vehicles per time unit that take part in the trip. A cost is associated with each edge to represent the travel time required to transverse it.

When such cost is modeled as a constant, the routing problem reduces to the Minimum-Cost Multi-Commodity Flow (MCMC) problem [7]. If fractional flows are allowed, this problem becomes a linear program and can be solved with traditional linear programming techniques.

On the other hand, to model the effect of congestion, the edge cost functions should be dependent on the current flow. In this work, we consider two popular flow-dependant cost functions:

- The Bureau of Public Roads (BPR) cost function [8]
- Greenshields' cost function [9, 10]

These two function will be analysed in detail in Section 4.2.1.

The introduction of flow-dependant costs gives rise to two different formulations for the routing problem [11]. The first is called **System Optimal (SO)** routing. It characterises the scenario where all users choose their paths in order to minimise the total system travel time or, in other worlds, the social cost. The second is known as the **User Equilibrium (UE)** or Wardrop equilibrium, from the seminal work in which Wardrop first introduced it [6]. The UE represents a scenario were routing is modeled as a non-cooperative game and users choose their paths greedily, converging to a Nash equilibrium. In this state, no user can improve its travel time by choosing a different path to its destination. UE routing is thus a self-interested routing formulation, in which every user chooses the most convenient road for himself. *In this work, we use the terms self-interested or selfish routing and UE routing interchangeably.*

SO routing is not fair to the users, as some user may be disadvantaged for the system's greater good. On the other hand, UE presents a fair solution in which every user going to the same destination experiences the same travel time, but it comes at the cost of system non-optimality. The cost trade-off in total travel time between these two scenarios is known as "The price of anarchy" [12, 13]. Both routing strategies can be casted as convex nonlinear optimisation problems, which can be solved by traditional gradient-based techniques such as the Frank-Wolfe algorithm [14, 4]. More on this in Section 5.1.2.

Given the unfairness resulting from SO, most of traffic assignment research has focused on UE routing. Apart from solving the optimisation problem, solutions have been proposed to compute UE paths that consider traffic at the microscopic vehicle level. A first solution computes the dynamic UE through iterative microscopic traffic simulations [15]. A second approach performs iterative traffic assignment by routing

vehicles online according to a global congestion view [3]. In this project, we use both of these works to solve UE routing in our simulations.

Example scenario

Let us consider the example shown in Figure 2.1 to illustrate the difference between SO and UE routing. In this simple scenario we consider only one trip, from s to t, with a demand of 1 unit of flow. In the network there are only two roads, represented as edges. The travel time of such roads is indicated by $c_1(x)$ and $c_2(x)$, which are functions of the flow x. The flow routed on the upper road is indicated as x_1, while the flow on the lower road is x_2.

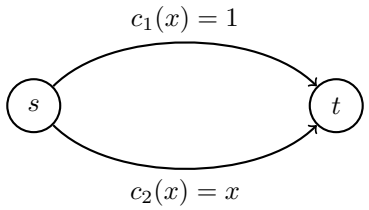


Figure 2.1: Simple transport network to illustrate the difference between UE and SO. It presents one origin s and one destination t and a flow demand of 1 that has to be distributed along paths 1 and 2, with travel time costs respectively $c_1(x)$ and $c_2(x)$.

If the users are routed according to UE, the flow will distribute such that all the users experience the same travel time. This leads to a UE where all the flow is routed on the lower road ($x_1 = 0, x_2 = 1$), yielding $c_1(x_1) = c_2(x_2) = 1$. The total system travel time is $x_1c_1(x_1) + x_2c_2(x_2) = 1$. On the other hand, if we optimise for the SO, the flow distributes equally on the two roads ($x_1 = 0.5, x_2 = 0.5$), yielding a total travel time of $x_1c_1(x_1) + x_2c_2(x_2) = 0.75$. This example is summarised in Table 2.1.

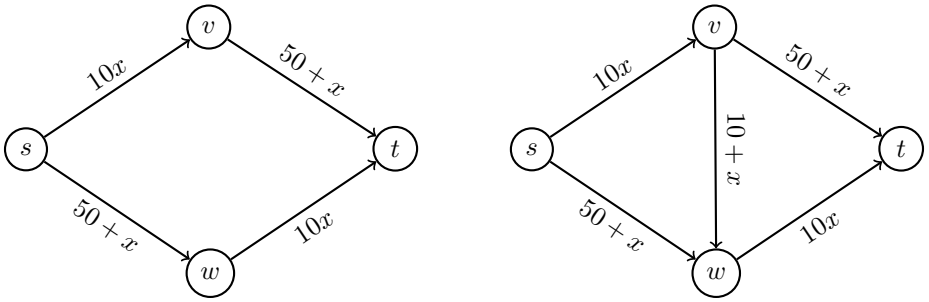
Table 2.1: UE and SO routing results on the simple network in Fig. 2.1.

Routing	x_1	x_2	Total travel time
User Equilibrium	0	1	1
System Optimal	0.5	0.5	0.75

2.2.2 Braess' paradox

One of the most popular results in traffic research is a routing paradox discovered by Braess [1] in 1968, which has since become known as the Braess' paradox. Braess' paradox states that, when users are routed according to UE, the total system travel time can increase after a new road is added to the network. We can see how this phenomenon is not really a paradox as, when users are behaving according to UE, they behave selfishly and thus probably do not make the best use of the transportation infrastructure available. Braess illustrates this through a simple yet somewhat pathological example. However, Braess' paradox has been shown to be commonplace in road networks [16, 17] and in large random networks [18].

In the following, we illustrate Braess' example.



(a) Before addition **(b)** After addition

Figure 2.2: Braess' paradox network. Labels on the edges represent the flow-dependant travel time functions.

Braess' paradox presents a proof that the total travel time of users routed in a network according to the UE is not monotonic non-increasing with respect to path additions. This is shown through a counterexample, reported in Figure 2.2. In this example, we consider a flow demand of 6 going from node s to node t. The road travel time cost functions are reported on the respective edges of the graph. In Figure 2.2a the solution to the SO and UE is simple: all the users spread equally on the available paths, leading to a flow of 3 on the top edges and a flow of 3 on the bottom edges. Every user experiences the same travel time of $50 + 3 + 10 * 3 = 83$. This solution is both optimal for the users and for the system. On the other hand, if the edge (v, w) is added to the network, as in Figure 2.2b, the solution to the SO does not change, while selfish users behaving according to the UE see an opportunity to decrease their travel time by using the newly created path $(s, v) \to (v, w) \to (w, t)$. Thus, when the equilibrium is reached we observe a flow of 2 in the previously available paths and a flow of 2 in the newly created one. In this UE scenario each user experiences the same travel time of 92. The increase in the total travel time of users behaving accordingly to UE after the addition of edge (v, w) is:

$$92 * 6 - 83 * 6 = 552 - 498 = 54$$

Braess' paradox informs us that extreme cation must be used when structuring transport networks. More precisely, two main insights can be derived. Firstly, particular attention has to be paid when considering adding new paths to road network, as it is possible that such additions may not only have no effect, but also damage the current efficiency of the network. Secondly, it tells us that, given a network, optimal solutions for the UE could be found by considering its subnetworks, that is, removing some edges could lead to great user benefits. These two insights respectively motivated the two main contributions of this project.

2.2.3 Traffic simulation

We also employ traffic simulations to validate our results. Traffic simulation models can be divided into three main categories [19]:

- **Macroscopic traffic modeling** [20], where traffic is simulated at a system level with traffic density represented as continuous flows in the road network.

- **Microscopic traffic modeling** [21] simulates traffic at a vehicle level. Each single vehicle can have different characteristics and parameters. This allows for detailed analyses on the interaction between different types of vehicles.
- **Mesoscopic traffic modeling** [22] bridges the two approaches by providing a trade-off between simulation complexity and level of detail.

In this work we will look into both microscopic traffic simulation and macroscopic flow simulation. After reviewing the available state of the microscopic traffic simulators [23, 24, 2] as well as literature surveys on the topic [25], we chose to use SUMO [2]. For the macroscopic flow simulation, we develop custom code.

SUMO

SUMO (Simulation Of Urban Mobility) [2] is a free and open-source traffic microsimulator. It provides interactive graphical tools for visualising simulations and creating road networks. Real-world networks can be imported from Open Street Map [26]. SUMO allows intermodal traffic by modelling pedestrians, public transport and cyclists.

2.3 Supermodularity

A fundamental component underlying the theoretical study in this dissertation is the property of supermodularity of set functions [27]. Before introducing such property, let us first define non-increasing monotonicity.

Definition 2.3.1 (**Monotonic non-increasing set function**)**.** Let Λ be a set function defined as $\Lambda : 2^{\mathcal{G}} \mapsto \mathbb{R}$, where $2^{\mathcal{G}}$ is the power set of the finite set $\mathcal{G}$. Λ is a monotonic non-increasing function of $\mathcal{G}$ if and only if for every $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{G}$, we have that $\Lambda(\mathcal{B}) \leqslant \Lambda(\mathcal{A})$.

In other words, if Λ is monotonic non-increasing, its output cannot increase when new elements are added to its current input. Let us now introduce supermodularity.

Definition 2.3.2. (**Supermodular set function**) Let Λ be a set function defined as $\Lambda : 2^{\mathcal{G}} \mapsto \mathbb{R}$, where $2^{\mathcal{G}}$ is the power set of the finite set $\mathcal{G}$. Λ is a supermodular function of $\mathcal{G}$ if and only if for every $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{G}$ and every $x \in \mathcal{G} \setminus \mathcal{B}$, we have that

$$\Lambda(\mathcal{A}) - \Lambda(\mathcal{A} \cup \{x\}) \geqslant \Lambda(\mathcal{B}) - \Lambda(\mathcal{B} \cup \{x\})$$

In other words, if Λ is supermodular and it represents some notion of cost, the utility of adding an element to its current input is always greater or equal than the utility of adding the same element to a superset of its current input.

It is fundamental to understand that none of these two properties implies the other. Therefore, a supermodular function can be non-monotonic.

2.4 Genetic algorithm

Metaheuristics are general algorithms designed to solve complex optimisation problems in a computational and time efficient manner. Unlike exact methods, they often do not

converge to exact solutions. However, exact solutions may be computationally infeasible to calculate and the trade-off between computational complexity and exactness is a fundamental factor to consider. The high level structure of metaheuristics enables them to be applied to a variety of optimisation problems.

Genetic algorithm (GA) [28] is a popular metaheuristic that belongs to the field of evolutionary computation [29], a class of optimisation algorithms inspired from biological evolution. It is, in fact, based on the Darwinian evolutionary theory, implementing the concept of "Survival of the fittest".

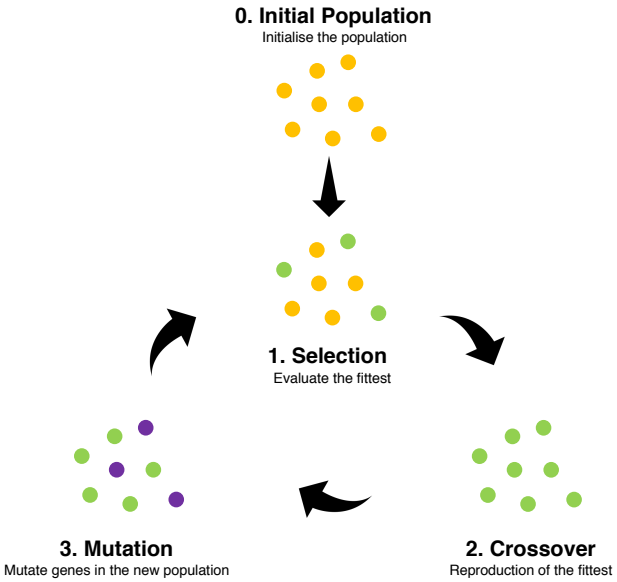


Figure 2.3: High level structure of the Genetic algorithm.

The general structure of this class of algorithms is depicted in Figure 2.3. It is divided in four main steps:

0) **Initial population:** In this phase a set of feasible solutions to the optimisation problem is created. Each solution in the population is composed by a set of attributes, called genes.

1) **Selection:** In this phase the fitness of each solution is computed according to a problem-dependent fitness function.

2) **Crossover:** Reproduction is carried out among the fittest individuals according to a given probability. The new generation is obtained by mixing the gene set of pairs of individuals.

3) **Mutation:** With a defined probability, each gene of each individual is subject to a modification called mutation. This introduces diversity in the current population.

This process continues until a certain stopping condition is met. The most important problem-dependant components that a user has to specify are: the fitness function, the crossover strategy, and the mutation function. By relying on probabilistic transition

rules, the GA is able in general to achieve better search space exploration with respect to traditional deterministic metaheuristics.

2.5 Reinforcement learning

Machine learning paradigms have seen an increasing popularity in recent years. Unlike traditional supervised learning settings, Reinforcement Learning (RL) [30] considers an agent autonomously interacting with the environment.

At each time step t, the autonomous agent observes a state $s_t \in \mathcal{S}$ that contains a representation of the environment. It then chooses an action $a_t \in \mathcal{A}$ based on the state observed. This action leads to the agent observing a new state $s_{t+1} \in \mathcal{S}$ and receiving a reward $r_{t+1} \in \mathbb{R}$, that encodes the goodness of its action. This interaction is illustrated in Figure 2.4.

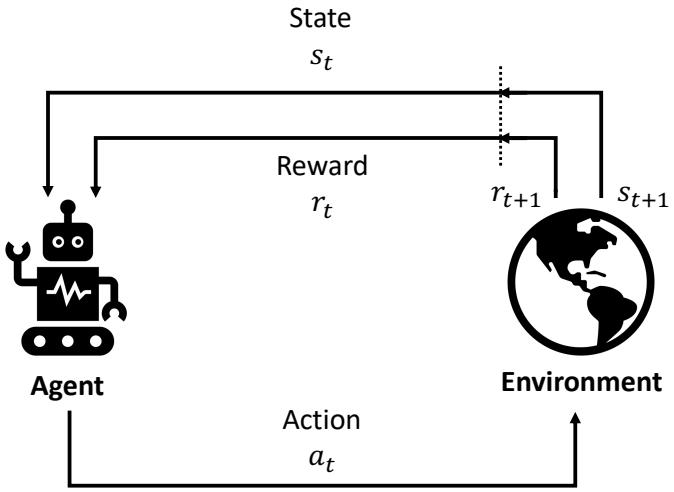


Figure 2.4: Reinforcement learning agent-environment interaction.

The sequential decision making task just described can be formulated as a Markov Decision Process (MDP). A MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$, where $\mathcal{S}, \mathcal{A}$ are the state and action spaces, $\mathcal{R}$ is a reward function $\mathcal{R} : \mathcal{S}, \mathcal{A} \mapsto \mathbb{R}$, $\mathcal{T}$ contains the state transition probabilities $\mathcal{T}(s_{t+1}|s_t, a_t)$ and γ is a reward discount factor where lower values place more emphasis on immediate rewards. A MDP is based on the Markov assumption, which informally states that "The future is independent from the past given the present". Therefore, in MDPs, the current state captures all the information from the agent interaction history.

The ultimate goal of the agent is to learn a policy π in order to maximise the total discounted reward. π is defined as an action distribution over the states $\pi(a|s) = P(a|s)$. The total discounted reward v_t, also called *return*, is defined as:

$$v_t = r_{t+1} + \gamma r_{t+2} + \cdots = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$$

Where T is the agent-environment interaction length, known as *episode length*. T can be infinite in the case of infinite length episodes.

Given a policy π it is possible to define the utility of each state. This is done using the *state-value* function:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[v_t | s_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \middle| s_t = s\right]$$

Which estimates the expected total discounted reward obtained if the agent starts in state s and follows policy π. We can also define the value of each action in each state through the *action-value* function:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[v_t | s_t = s, a_t = a] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \middle| s_t = s, a_t = a\right]$$

The best possible agent performance in the MDP is then obtained by maximising these functions.

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$
$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

$V^*(s)$ and $Q^*(s,a)$ are the state-value function and the action-value function corresponding to the optimal policy π^*. Note that all optimal policies obtain these value functions and there always exists a deterministic optimal policy for any MDP.

Many iterative methods have been proposed for finding the optimal policy. While methods like dynamic and linear programming can reach the optimum in simple scenarios, they become infeasible for complex MDPs. RL approaches like Monte Carlo methods, Temporal Difference learning, Q-learning and SARSA have been used in the RL literature. Recently, Deep Neural Networks (DNN) have been shown to be provide effective solutions [31].

Proximal policy optimisation

In this work we use Proximal Policy Optimisation (PPO) [32], a deep RL [33] algorithm for finding the optimal policy.

PPO aims to compute a stochastic parametric policy π_θ by employing two neural networks: the *actor* and the *critic*. The critic tries to approximate the state-value function for each state and the actor encodes the policy π_θ. The policy is updated through gradient descent based on a loss obtained by comparing these two components. The algorithm is *model-free* meaning that it learns the MDP model dynamics through agent-environment interactions. The innovative aspect of PPO with respect to other policy gradient methods is that it avoids large policy updates by simply clipping the loss.

Chapter 3

Related work

With the introduction of Autonomous Vehicles (AVs) we are facing a societal shift towards new mobility systems based on driverless vehicle control. These technologies are enabling new paradigms such as Autonomous Mobility-On-Demand [34]. The modeling of vehicles as flow in a capacitated network has been recently studied in these systems [35]. A key component that has to be taken into consideration is the environment and its impact on AVs navigation [36]. A line of research focuses on the co-design of mobility systems [37]. Recent work in multi-robot task assignment [38] has shown interesting correlation patterns between the assignment procedure and the diversity in the paths chosen by the robots traversing the transport network. Given the high potential and impact that environment optimisation can have on these robotics systems, in this work we aim to gain general insights and results on the problem by aligning with related work in traffic research.

The discovery of Braess' paradox [1] advised researchers that extra care had to be put in designing transport networks. Seminal works have analysed the trade-off between User Equilibrium (UE) and System Optimal (SO) routing [11]. The impact of network topology [13] and the impact of centrally controlled users [39] on routing have been investigated as well. These works led to a hardness result regarding the problem of designing networks for selfish users [40], exacerbating the severity of Braess' paradox. For this reason, several works investigated how Braess' paradox could be detected and avoided: [41] focuses on the hard problem of identifying optimal "Braess-free" sub-networks, [42] analyses the range of trip demands that do not cause the paradox, [43] proposes an heuristic solution to identify the road segments causing traffic delays.

Another approach, close to ours, to incentivise selfish users to make optimal use of the transport network, is the introduction of road pricing policies, implemented through road tolls [44]. This strategy implements an economic model of route guidance where users are penalised economically proportionally to the non-optimality of their behaviour. [45] analyses the benefit of such pricing policies in networks with linear latency functions and reaches the conclusion that taxes in such networks are at most as effective as edge removals.

The central problem in traffic research that focuses on network design is known as the Network Design Problem (NDP). It is formulated as a bilevel optimisation task. At the higher level, the network designer can make decisions that impact the network

infrastructure, these decisions can be discrete (adding and removing edges) or continuous (extending the edge capacities). Decisions involve physical network modifications and are subject to a budget. At the lower level, users see the network modifications and route according to the UE on the modified network. Note that the designer cannot control directly the users' behaviour, but only influence it through environment optimisation. This kind of setting is also known as a Stackelberg game [46]. The designer goal is to minimise the total system travel time while respecting the construction budget. In other words, make the best use of infrastructure improvements in order to make selfish users indirectly behave optimally. Our task is extremely related to this, but we do not consider extending the current infrastructure, instead we leverage insights from Braess' paradox to make some links look less appetible to selfish users, not through a pricing scheme, but through a modified view of the network in which the capacity of some links has been decreased.

For a comprehensive review of the NDP, see [47]. The discrete formulation of the problem was first introduced by LeBlanc in 1975 [48], where he solves this NP-Hard problem by utilising the branch and bound algorithm and a clever SO relaxation for the lower bounds. This is a computationally infeasible solution for large networks and, thus, heuristic approaches have been proposed [49, 50]. The Genetic Algorithm (GA) represents one effective metaheuristic solution [51]. The continuous version of the problem was introduced in [52] and still constitutes an hard optimisation task due to the non-convexity deriving from the bilevel formulation. Also in this case, metaheuristics such as simulated annealing [53] and GA [54, 55] have shown to perform nearly optimal.

Chapter 4

Supermodularity of path additions

In this chapter, we analyse the impact of the addition of paths to a transport network where Autonomous Vehicles (AVs) are routed. We tackle this problem from a theoretical perspective, in order to investigate the formal properties of the utility gained from extending the transport network. We prove that the intuitive statement *adding a path to a road network always grants greater or equal benefit to road users than adding it to a bigger network*[1] is false.

In the first part of this chapter (Sec. 4.1), we model the problem of routing AVs in congested networks as a classical minimum cost multi-commodity network flow problem [7]. We then move to defining the concepts of *trip spanning tree* and *path addition* which will allow us to formalise how we extend the transport network. Then, we focus on the routing objective function and, in particular, on the effects that extending the input graph has on the routing cost. We show that the rather intuitive property of supermodularity of path additions to transport networks does not hold in the general multi-commodity case. We also prove that, in the more specific case of parallel paths with arbitrary costs and fixed capacities, we obtain diminishing returns upon the addition of new paths.

In the second part (Sec. 4.2), we introduce the problem of time delays due to congestion by extending the multi-commodity network flow problem to the case of flow-dependant costs. We show that also in this case, for both System Optimal (SO) and User Equilibrium (UE) routing, the rather intuitive property of supermodularity of path additions to transport networks does not hold. We then analyse the impact of network extension in a specific routing scenario where we prove that path additions produce a supermodular improvement in routing.

The chapter is structured in the following way: Section 4.1.1 introduces the minimum cost network flow problem and explains how such problem is extended for multiple AVs trips, Section 4.1.2 defines how we model the initial transport network and its extensions, Section 4.1.3 investigates the supermodularity of path additions in congestion-agnostic routing, Section 4.2.1 introduces the notion of flow-dependant costs, Section 4.2.2 and 4.2.3 characterise the SO and UE problems, Section 4.2.4 analyses the differences between

[1] A superset of the original network

these two problems, in Section 4.2.5 we investigate the impact that network extensions have on routing performance in this scenario, Section 4.2.6 proves the supermodularity of path additions for identical parallel paths and, finally, Section 4.3 summarises the chapter.

4.1 Congestion-agnostic routing

4.1.1 Minimum cost network flow problem

The minimum cost network flow problem (MC) is a classic flow problem [7]. It is concerned with capacitated networks in which a cost is associated to every edge. Given a source node s, a sink node t, and a flow demand d, we are interested in finding the optimal flow distribution in the capacitated network such that the total cost is minimised. The shortest path problem and the maximum flow problem are special cases of the MC problem. In this work, we treat the flow source and sink as origin and destination of a AVs trip, where the demand represents the number of vehicles per time unit going from that origin to that destination.

Given an oriented graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the node set and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the edge set, we associate to each node a demand $d_i \in \mathbb{R}, \forall i \in \mathcal{V}$ and to each edge a unitary cost $c_{ij} \in \mathbb{R}$ and a capacity $u_{ij} \in \mathbb{R}$, $\forall (i,j) \in \mathcal{E}$. Note that, in this formulation, we allow $c_{ij} \neq c_{ji}$. We define a trip as a triple (s, t, d), where $s \in \mathcal{V}$ is the origin node, $t \in \mathcal{V}$ is the destination node and d is the vehicle demand. We set the network demands d_i as indicated by Equation 4.1.

$$\forall i \in \mathcal{V} \quad d_i = \begin{cases} -d & \text{if } i = s \\ d & \text{if } i = t \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

We observe that the total demand of the network is equal to zero $\sum_{i \in \mathcal{V}} d_i = 0$.

We denote the vehicle flow routed on an edge as $x_{ij}, \forall (i,j) \in \mathcal{E}$. The routing problem consists in determining the optimal vehicle distribution on the network edges in order to route all the demand d from s to t while minimising the total cost. The problem is formalised as follows

$$\min_{\mathbf{x}} \sum_{(i,j) \in \mathcal{E}} c_{ij} x_{ij} \tag{4.2a}$$

s.t.

$$0 \leqslant x_{ij} \leqslant u_{ij} \quad \forall (i,j) \in \mathcal{E} \tag{4.2b}$$

$$\sum_{\{j:(j,i) \in \mathcal{E}\}} x_{ji} - \sum_{\{j:(i,j) \in \mathcal{E}\}} x_{ij} = d_i \quad \forall i \in \mathcal{V} \tag{4.2c}$$

The objective function 4.2a minimises the total cost of the routing. Constraint 4.2b ensures that the vehicle flow on each edge is non-negative and does not exceed the maximum capacity and constraint 4.2c ensures flow conservation.

Minimum cost multi-commodity network flow problem

Now that we are familiar with the MC problem, we can introduce the minimum cost multi-commodity network flow problem (MCMC). This is a generalisation of the problem just described where we want to model heterogeneous commodities from different origins and destinations. In our application this means that we can define multiple trips with different origins, destinations, and demands. Each trip will hence correspond to a commodity. *From this point on we will use the terms commodity and trip interchangeably.*

We define the set of trips as $\mathcal{M} = \{(s^m, t^m, d^m)\}$ and the demand at each node as

$$\forall i \in \mathcal{V},\ \forall m \in \mathcal{M} \quad d_i^m = \begin{cases} -d^m & \text{if } i = s^m \\ d^m & \text{if } i = t^m \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

We represent the vehicle flow of trip $m \in \mathcal{M}$ on edge $(i,j) \in \mathcal{E}$ as x_{ij}^m. The minimum cost multi-commodity flow problem can then be then defined as follows

$$\min_{\mathbf{x}} \sum_{m \in \mathcal{M}} \sum_{(i,j) \in \mathcal{E}} c_{ij} x_{ij}^m \tag{4.4a}$$

s.t.

$$\sum_{m \in \mathcal{M}} x_{ij}^m \leqslant u_{ij} \quad \forall (i,j) \in \mathcal{E} \tag{4.4b}$$

$$x_{ij}^m \geqslant 0 \quad \forall (i,j) \in \mathcal{E},\ \forall m \in \mathcal{M} \tag{4.4c}$$

$$\sum_{\{j:(j,i) \in \mathcal{E}\}} x_{ji}^m - \sum_{\{j:(i,j) \in \mathcal{E}\}} x_{ij}^m = d_i^m \quad \forall i \in \mathcal{V},\ \forall m \in \mathcal{M} \tag{4.4d}$$

Where the objective function 4.4a minimises the total cost of routing over all the trips. Constraint 4.4b ensures that the sum of the flows of different trips on each edge does not exceed the maximum capacity. Constraint 4.4c ensures that no flow is negative. Constraint 4.4d ensures flow conservation.

Path formulation

According to the flow decomposition theorem, first introduced by [56], we can reformulate the MCMC problem using the distribution of flow over paths instead of edges. This is valid under the assumption that for every trip there exist no negative cost cycles in our graph.

For every trip $m \in \mathcal{M}$ let P^m denote the set of all possible paths[2] from s^m to t^m. The cost c_p of a path is defined as the sum of the cost of its edges $c_p = \sum_{(i,j) \in p} c_{ij}$. We also define the total flow of trip m on path $p \in P^m$ as x_p. The path formulation goes as follows

$$\min_{\mathbf{x}} \sum_{m \in \mathcal{M}} \sum_{p \in P^m} c_p x_p \tag{4.5a}$$

[2]A path is a is a walk in which all vertices (and therefore also all edges) are distinct.

s.t.

$$\sum_{m \in \mathcal{M}} \sum_{p \in P^m : (i,j) \in p} x_p \leqslant u_{ij} \quad \forall (i,j) \in \mathcal{E} \tag{4.5b}$$

$$x_p \geqslant 0 \quad \forall p \in P^m, \ \forall m \in \mathcal{M} \tag{4.5c}$$

$$\sum_{p \in P^m} x_p = d^m \quad \forall m \in \mathcal{M} \tag{4.5d}$$

Where the objective function and the constraints map exactly to the ones illustrated in the edge MCMC formulation.

4.1.2 Trip spanning tree and path additions

Now that we have introduced the routing problem, we turn our attention to the transport network on which vehicles are routed, represented as a graph.

We start by defining a set of properties which will be common to all the graphs we will consider. The graphs we will describe will have a cost $c_{ij} > 0$ and a capacity $u_{ij} \geqslant 0$ associated to each edge. Furthermore, they will all be subgraphs of a template graph $G_{\mathcal{T}}$, used to limit the space of possible graph extensions.

We start by introducing the concept of *trip spanning tree*.

Definition 4.1.1 (**Trip spanning tree**)**.** Given a set of commodities $\mathcal{M} = \{(s^m, t^m, d^m)\}$, a trip spanning tree is a directed graph $G_{\mathcal{I}} = (\mathcal{V}_{\mathcal{I}}, \mathcal{E}_{\mathcal{I}})$ with the following four properties:

1. $s^m, t^m \in \mathcal{V}_{\mathcal{I}}, \ \forall m \in \mathcal{M}$
2. $|P^m| = 1, \ \forall m \in \mathcal{M}$
3. $\forall n \in \mathcal{V}_{\mathcal{I}}, \ \exists m \in \mathcal{M}, \ p \in P^m : n \in p$
4. $\sum_{m \in \mathcal{M}} \sum_{p \in P^m : (i,j) \in p} d^m \leqslant u_{ij} \quad \forall (i,j) \in \mathcal{E}_{\mathcal{I}}$

Property (1) states that a trip spanning tree must contain the source and destination of every trip. Property (2) states that for each origin-destination pair there exists exactly one path connecting them in the trip spanning tree. Property (3) states that every node in the trip spanning tree must be part of a path connecting one trip's origin to its destination. Property (4) states that the capacity of the paths in the trip spanning tree must be sufficient to guarantee a feasible solution to the MCMC problem.

We now introduce the concept of a commodity path graph.

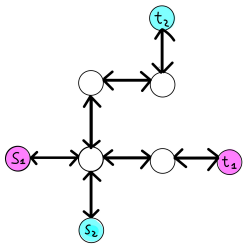
Definition 4.1.2 (**Commodity path graph**)**.** We denote a commodity path graph, with respect to commodity $m \in \mathcal{M}$, as a graph $G_x = (\mathcal{V}_x, \mathcal{E}_x)$ with the following properties:

1. $s^m, t^m \in \mathcal{V}_x$
2. $|P^m| = 1$
3. $\forall n \in \mathcal{V}_x, \ p \in P^m : n \in p$

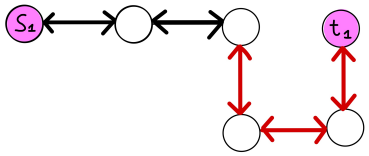
A commodity path graph is thus a path graph[3] which contains only a path from one trip source to its destination. Property (1) states that the origin and destination of such trip must belong to the commodity path graph. Property (2) states that the commodity path graph contains only one path between these two. Property (3) states that all the nodes belonging to the commodity path graph must belong to such path. Therefore, a commodity path graph is a trip spanning tree with respect to only one commodity, without the guarantees given by Property (4) of a trip spanning tree.

Given a set of trips and a trip spanning tree for this set, we can model the subsequent addition of commodity path graphs to the trip spanning tree as a graph union. The graph obtained by adding a commodity path graph $G_x = (\mathcal{V}_x, \mathcal{E}_x)$ to the trip spanning tree $G_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{E}_\mathcal{I})$ is described as $G_{\mathcal{I}\oplus x} = (\mathcal{V}_\mathcal{I} \cup \mathcal{V}_x, \mathcal{E}_\mathcal{I} \cup \mathcal{E}_x)$. Note that, by this addition, the commodity path graph can introduce multiple new nodes and edges not previously contained in the trip spanning tree and thus also multiple paths for each commodity. The number of new paths introduced is limited only by the graph template $G_\mathcal{T}$ (remembering that $G_\mathcal{I} \subseteq G_\mathcal{T}$ and $G_x \subseteq G_\mathcal{T}$).

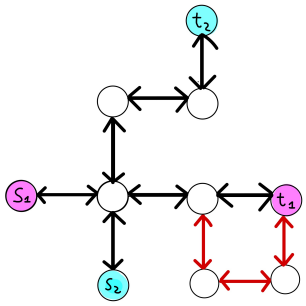
We illustrate the concept of trip spanning tree and commodity path addition with an example, shown in Figure 4.1. In this simple example we have only two trips $\mathcal{M} = \{(s^1, t^1, d^1), (s^2, t^2, d^2)\}$. We instantiate a possible trip spanning tree (4.1a) and a possible commodity path graph (4.1b) for trip 1. The resulting graph union $G_{\mathcal{I}\oplus x}$ is shown in 4.1c. Note that in this case G_x adds one path for trip 1 and none for trip 2, but this may not always be the case.



(a) $G_\mathcal{I}$



(b) G_x



(c) $G_{\mathcal{I}\oplus x}$

Figure 4.1: A simple transport network example to illustrate graph unions. Here we have a trip spanning tree ($G_\mathcal{I}$) and a commodity path graph (G_x) that are being unified into $G_{\mathcal{I}\oplus x}$.

[3] `https://en.wikipedia.org/wiki/Path_graph`

Let us introduce some further notation related to graph unions. Let us take a graph $G_{\mathcal{A}} = (\mathcal{V}_{\mathcal{A}}, \mathcal{E}_{\mathcal{A}})$, obtained after $n \geqslant 0$ commodity path graph additions to a trip spanning tree, and a commodity path graph $G_x = (\mathcal{V}_x, \mathcal{E}_x)$. We denote with $P^m_{\mathcal{A}}$ the set of all paths for commodity m in $G_{\mathcal{A}}$, with $P^m_{x|\mathcal{A}}$ the set of all paths added by G_x to $G_{\mathcal{A}}$ for commodity m, and with $P^m_{x\oplus\mathcal{A}}$ the set of all paths for commodity m in $G_{x\oplus\mathcal{A}}$. Note that $P^m_{x|\mathcal{A}}$ is defined as $P^m_{x|\mathcal{A}} = P^m_{x\oplus\mathcal{A}} \setminus P^m_{\mathcal{A}}$. From this definition we obtain that $P^m_{x\oplus\mathcal{A}} = P^m_{x|\mathcal{A}} \cup P^m_{\mathcal{A}}$, with $P^m_{x|\mathcal{A}} \cap P^m_{\mathcal{A}} = \varnothing$. It is important to note that the set $P^m_{x|\mathcal{A}}$ will always be a super set of P^m_x ($P^m_x \subseteq P^m_{x|\mathcal{A}}$). We also define $P_{\mathcal{A}} = \bigcup_{m\in\mathcal{M}} P^m_{\mathcal{A}}$.

Let us look at another example to exemplify this notation. Suppose that there exists only one trip $\mathcal{M} = \{(s^1, t^1, d^1)\}$. In Figure 4.2, we can see a trip spanning tree $G_{\mathcal{I}}$ for trip 1 (shown in black) and a commodity path graph G_x for the same trip (shown in red). Therefore, we will have $|P^1_{\mathcal{I}}| = |P^1_x| = 1$. These two sets will contain the black and the red path respectively. The set $P^1_{x\oplus\mathcal{I}} = P^1_{x|\mathcal{I}} + P^1_{\mathcal{I}}$ will contain four paths: three of which come from the set $P^1_{x|\mathcal{I}}$ and the last being the original path present in $P^1_{\mathcal{I}}$.

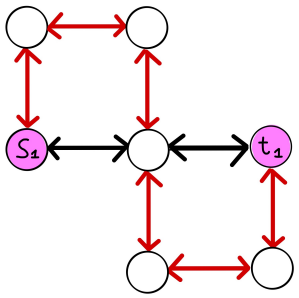


Figure 4.2: Simple transport graph to illustrate the addition of a commodity path graph. In this figure we have a trip spanning tree $G_{\mathcal{I}}$ for trip 1 (shown in black) to which we add a commodity path graph G_x for the same trip (shown in red). We see how, by only adding one commodity path graph to the trip spanning tree, we obtain four total paths for trip 1.

4.1.3 Properties of path additions

Given a finite set of commodity path graphs $\mathcal{G}$, we define the function $\Lambda : 2^{\mathcal{G}} \mapsto \mathbb{R}$ which takes as input a subset of $\mathcal{G}$ and adds it to a fixed trip spanning tree defined over a fixed set of commodities and outputs the value of cost function 4.5a on the resulting graph, while following constraints 4.5b, 4.5c, and 4.5d.

Let's take as an example $\mathcal{A} \subseteq \mathcal{G}$ which is a set of commodity path graphs. For our purpose this set is equally identified by the union of all the graphs in it, which we call $G_{\mathcal{A}} = \bigcup_{G\in\mathcal{A}} G$ which is also identified by the set of its commodity paths $P_{\mathcal{A}}$. Therefore we can write

$$\Lambda(\mathcal{A}) = \min_{\mathbf{x}} \left\{ \sum_{m\in\mathcal{M}} \sum_{p\in P^m_{\mathcal{A}\oplus\mathcal{I}}} c_p x_p \right\} \tag{4.6}$$

where $P^m_{\mathcal{A}\oplus\mathcal{I}}$ represents the set of paths for commodity m in graph $G_{\mathcal{A}\oplus\mathcal{I}}$ where $G_{\mathcal{I}}$ is the fixed trip spanning tree.

Note that we need the trip spanning tree $G_{\mathcal{I}}$ to guarantee that even when Λ is computed on the empty set ($\Lambda(\varnothing)$) there is still a feasible flow for each trip and each

trip source its connected to its destination.

Theorem 4.1. *Given two sets of commodity path graphs $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{G}$, we have that $\Lambda(\mathcal{B}) \leqslant \Lambda(\mathcal{A})$, thus Λ is a monotonic non-increasing function of the set $\mathcal{G}$.*

Proof. For brevity, the proof can be found in Appendix A.1. ■

Now, we want to see if Λ is supermodular on the set $\mathcal{G}$. To prove it, we need to show that: given two subsets $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{G}$ and $x \in \mathcal{G} \setminus \mathcal{B}$, it holds that

$$\Lambda(\mathcal{A}) - \Lambda(\mathcal{A} \cup \{x\}) \geqslant \Lambda(\mathcal{B}) - \Lambda(\mathcal{B} \cup \{x\}) \tag{4.7}$$

Theorem 4.2. *Λ is **not** a supermodular function of the set the set $\mathcal{G}$.*

Proof. We show that supermodularity does not hold by finding a counterexample. Furthermore, the counterexample found is valid also in the case of a single commodity. Therefore, Λ is proven not to be supermodular even in the simple MC problem. We will now illustrate the counterexample.

The example is illustrated in Figure 4.3. It is characterised by the following data:

- There is only one trip $\mathcal{M} = \{(s^1, t^1, 5)\}$ with a demand of 5
- Every edge has a capacity of 5
- Edges shown in black have a cost of 2
- All other edges have a cost of 1

We take $\mathcal{A} = \varnothing$, $x = G_x$, and $\mathcal{B} = \mathcal{Y} = G_{\mathcal{Y}}$ as shown in Figure 4.3.
Equation 4.7 becomes

$$\Lambda(\varnothing) - \Lambda(\{x\}) \geqslant \Lambda(\mathcal{Y}) - \Lambda(\mathcal{Y} \cup \{x\}) \tag{4.8}$$

Which we rewrite as

$$\begin{aligned} &\min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{\mathcal{I}}} c_p x_p \right\} - \min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{x \oplus \mathcal{I}}} c_p x_p \right\} \geqslant \\ &\min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{\mathcal{Y} \oplus \mathcal{I}}} c_p x_p \right\} - \min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{x \oplus \mathcal{Y} \oplus \mathcal{I}}} c_p x_p \right\} \end{aligned} \tag{4.9}$$

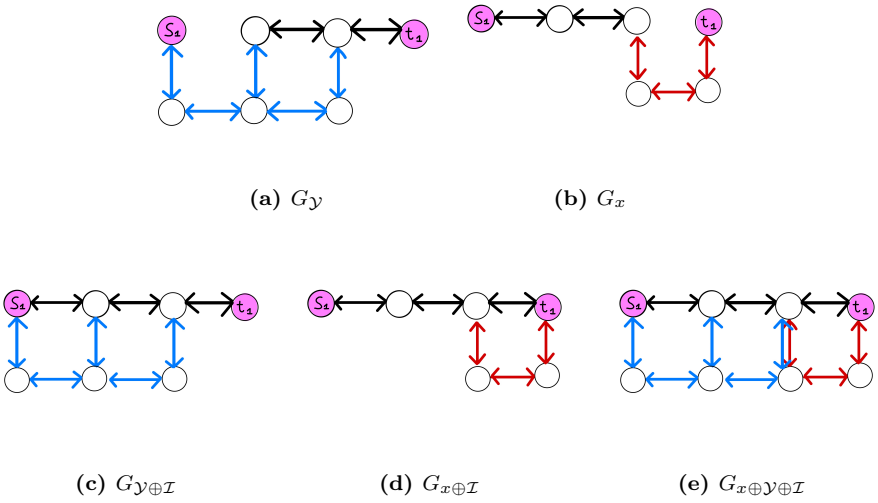
Which when solved reduces to

$$30 - 30 \geqslant 30 - 25 \tag{4.10}$$

Proving that Λ is not a supermodular function of the set $\mathcal{G}$. ■

It is now proven that in the general case of commodity path graphs that can have common edges and nodes, Λ is not supermodular.

Research question. Is Λ supermodular when the added commodity path graphs paths are allowed to have common nodes but not common edges? In other words, we restrict the set $\mathcal{G}$ to $\mathcal{G}' = \{x | x \in \mathcal{G} \wedge \forall y \in \mathcal{G},\ y \neq x \rightarrow \mathcal{E}_x \cap \mathcal{E}_y \cap \mathcal{E}_{\mathcal{I}} = \varnothing\}$. Is Λ a supermodular set function of $\mathcal{G}'$?



(a) $G_{\mathcal{Y}}$ **(b)** G_x

(c) $G_{\mathcal{Y}\oplus\mathcal{I}}$ **(d)** $G_{x\oplus\mathcal{I}}$ **(e)** $G_{x\oplus\mathcal{Y}\oplus\mathcal{I}}$

Figure 4.3: Counterexample of supermodularity when added commodity path graphs are allowed to have common nodes and edges.

Theorem 4.3. Λ *is* ***not*** *a supermodular function of the set the set* $\mathcal{G}'$*.*

Proof. Also in this case the answer is negative. We show this result through a counterexample.

Let us consider the graph depicted in Figure 4.4 where $G_{\mathcal{I}}$ is shown in black, $G_{\mathcal{Y}}$ is shown in blue, and G_x is shown in red. We still consider $\mathcal{A} = \varnothing$, $\mathcal{B} = \mathcal{Y} = G_{\mathcal{Y}}$ and $x = G_x$, thus, Equation 4.8 still holds. This example is characterised by the following:

- There is only one trip $\mathcal{M} = \{(s^1, t^1, 1)\}$ with a demand of 1
- Every edge has a capacity $\geqslant 1$
- Edges shown in black have a cost of 3
- All other edges ave a cost of 1

When solving Equation 4.9 we now obtain

$$15 - 13 \geqslant 13 - 7 \tag{4.11}$$

Which does not hold, proving that Λ is not supermodular on $\mathcal{G}'$. ■

4.1.4 Parallel paths

Having proven that Λ is not supermodular on $\mathcal{G}'$, we now consider the case of parallel commodity path graphs. In particular, we also restrict our attention to the single commodity case (MC problem), as, due to the parallel nature of the paths we consider, each commodity can be treated independently.

We thus restrict the space of possible commodity path graph additions to $\mathcal{G}'' = \{x | x \in \mathcal{G}' \wedge \forall y \in \mathcal{G}',\, y \neq x \rightarrow \mathcal{V}_x \cap \mathcal{V}_y \cap \mathcal{V}_{\mathcal{I}} = \{s, t\}\}$ where s, t are respectively the origin and destination of the considered commodity (s, t, d).

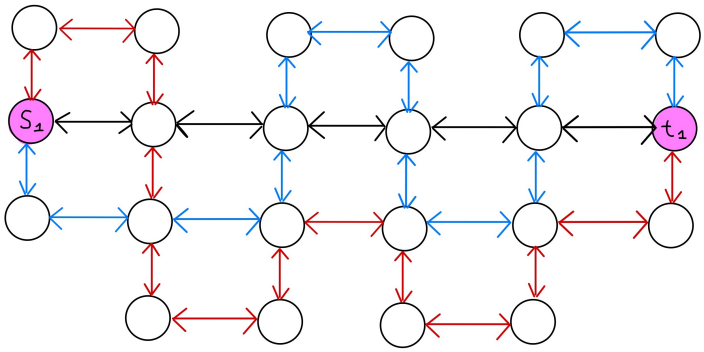


Figure 4.4: Counterexample of supermodularity when added commodity path graphs are allowed to have common nodes but not common edges.

The graphs considered in this section are obtained after multiple additions (taken from $\mathcal{G}''$) to a trip spanning tree. They will be represented using two nodes: origin (s) and destination (t), and a set of directed edges, representing the parallel paths connecting s with t. Each path has a cost $c_p = \sum_{(i,j)\in p} c_{ij}$ and a capacity $u_p = \min_{(i,j)\in p} u_{ij}$. An example of such network can be seen in Figure 4.5.

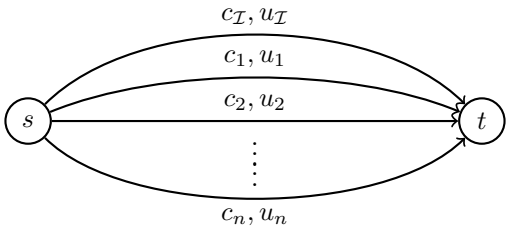


Figure 4.5: An example of a graph in the single commodity parallel paths case. Here we can see the trip spanning tree $G_\mathcal{I}$ (which in this scenario is always consisting of just one path with cost $c_\mathcal{I}$ and capacity $u_\mathcal{I}$) and some commodity path graphs numbered from 1 to n. The notation on the edges shows cost and capacity of each path, separated by a comma.

In the remaining of this section, we prove that Λ is supermodular on $\mathcal{G}''$ when it holds that $\forall x \in \mathcal{G}''\ u_x \geqslant d$, meaning that each possible addition is able to sustain the entire flow demand d.

In this scenario, the MC assignment becomes trivial, as the best solution is always to route the whole flow demand on the minimum cost path available in the graph. Hence, we can rewrite 4.5a as follows

$$\min_{\mathbf{x}} \sum_{m\in\mathcal{M}} \sum_{p\in P^m} c_p x_p = \min_{\mathbf{x}} \sum_{p\in P} c_p x_p = d \min\{c_p | p \in P\} \tag{4.12}$$

Lemma 4.4. *Given the sets of paths $P_\mathcal{A}$ and $P_\mathcal{B}$ obtained respectively from graphs $G_\mathcal{A}, G_\mathcal{B}$, we have that $P_\mathcal{A}, P_\mathcal{B} \subseteq P_{\mathcal{A}\oplus\mathcal{B}}$.*

Proof. The proof follows from the fact that graph $G_{\mathcal{A}\oplus\mathcal{B}}$ is defined as $G_{\mathcal{A}\oplus\mathcal{B}} = G_\mathcal{A} \cup G_\mathcal{B}$ and thus we can write $P_\mathcal{A} \cup P_\mathcal{B} \subseteq P_{\mathcal{A}\oplus\mathcal{B}}$. Hence, we can rewrite the thesis of Lemma 4.4 as $P_\mathcal{A}, P_\mathcal{B} \subseteq P_\mathcal{A} \cup P_\mathcal{B}$ which is true by the definition of the union over sets. ■

Theorem 4.5. *Given the set of possible parallel path additions $\mathcal{G}''$ and the function Λ, if $\forall x \in \mathcal{G}''\ u_x \geqslant d$ then Λ is a supermodular function of the set $\mathcal{G}''$.*

Proof. For brevity, the proof can be found in Appendix A.2. ■

The proof heavily depends on the fact that each path can sustain the entire flow. Therefore, the addition of a new path can either:

- Make no change (because it has a higher cost than the best path already in the graph) and will continue to make no change when considering adding it to a superset of the graph.
- Make an improvement (because it is the new best path). The improvement is its cost minus the previous best cost. Adding this path to superset of the graph will at best make the same improvement (if the superset graph has the same best cost) or a smaller improvement (if the superset graph has a better best cost than previously).

4.2 Congestion-aware routing

4.2.1 Flow-dependant cost

Up to this point we have investigated the impact of path additions in the traditional formulations of the MCMC and MC problems. This is a good starting point to theoretically characterise the problem at hand, but, in reality, it is often not possible to map the travel time of road segments to a constant cost. In fact, *the travel time required to traverse a road is highly dependent on traffic congestion.* This dependence has been observed and characterised since early traffic studies [9].

To model this dependence we will use the fundamental diagram of traffic flow, introduced by Greenshields [9, 10]. The cost of an edge will be modeled by a function $c_e : [0, u_e] \mapsto \mathbb{R}^+$, that takes as input the amount of flow x_e routed on edge e and outputs the cost. It is defined as follows

$$c_e(x) = \frac{l_e}{v_e^{max}\left(1 - \frac{x}{u_e}\right)} \tag{4.13}$$

Where v_e^{max} is the maximum velocity allowed on edge e and l_e is the length of e. Note that the input variable x_e has to be a feasible assignment and thus $0 \leqslant x_e \leqslant u_e$.

We chose this formulation as it is known to be one of the simplest yet most effective models for flow-dependant cost.

We now introduce Lemma 4.6 which we will use in later sections.

Lemma 4.6. *The function c_e is convex.*

Proof. The proof is trivial and can be found in Appendix A.3 ■

Another frequently considered latency function is the The Bureau of Public Roads (BPR) cost function [8]. The function is computed as:

$$c_e^{\mathrm{BPR}}(x) = c_e^0\left(1 + \alpha\left(\frac{x}{u_e}\right)^{\beta}\right) \tag{4.14}$$

where c_e^0 and u_e are the free-flow time and capacity of link e, respectively, and α and β are shape parameters which can be calibrated to data. In this work we use $\alpha = 0.15$ and $\beta = 4$. Although this function is not used in this chapter, all the insights we gain remain valid also for this latency function.

4.2.2 System optimal routing

Now we can use the cost function just introduced to extend our formulation of the MCMC problem. We briefly recall the relevant notation. We consider a graph $G = (\mathcal{V}, \mathcal{E})$ and a set of commodities $\mathcal{M} = \{(s^m, t^m, d^m)\}$. We denote the set of paths connecting s^m to t^m as P^m and define $P = \bigcup_{m \in M} P^m$. Each edge $e \in \mathcal{E}$ is assigned the flow-dependant cost function c_e introduced in the previous section. The vector of edge cost functions c_e is noted as $\mathbf{c}$. We call the triple $(G, \mathcal{M}, \mathbf{c})$ an instance.

The cost of a path $p \in P$ with respect to a flow x is defined as the sum of the cost of the edges in the path $c_p(x) = \sum_{e \in p} c_e(x)$

The optimal flows are then characterised by the following System Optimal (SO) routing problem

$$\min_{\mathbf{x}} \sum_{p \in P} x_p c_p(x_p) \tag{4.15a}$$

s.t.

$$x_p \geqslant 0 \quad \forall p \in P \tag{4.15b}$$

$$\sum_{p \in P^m} x_p = d^m \quad \forall m \in \mathcal{M} \tag{4.15c}$$

Where constraint 4.15b states that all flows must be non-negative and constraint 4.15c imposes the conservation of flow for each commodity.

Note that the objective 4.15a can be also written as

$$\min_{\mathbf{x}} \sum_{e \in \mathcal{E}} x_e c_e(x_e) \tag{4.15d}$$

But, in this case, we are forced to introduce an extra constraint imposing that the flow on an edge is equal to the sum of all the flows of the paths that pass through it:

$$x_e = \sum_{p \in P : e \in p} x_p \quad \forall e \in \mathcal{E} \tag{4.15e}$$

Since the local and global optima of a convex function on a convex set coincide [57], we now that a locally optimal solution for SO is also globally optimal whenever the objective function 4.15a is convex. By Lemma 4.6 we know that this is true for c_e and thus for c_p since a non-negative weighted sum of convex functions is still convex.

We know that a flow is locally (and globally) optimal if and only if moving flow from one path to another can only increase the global cost 4.15a. That is, we expect to have optimality when the marginal benefit of decreasing flow on a $s^m - t^m$ path is at most the marginal cost of increasing flow on another $s^m - t^m$ path [11]. Let us denote

$k_e(x) = x_e c_e(x)$. We write the derivative of k_e as k'_e and define $k'_p(x) = \sum_{e \in p} k'_e(x)$. We then have

Lemma 4.7 ([58, 59, 11]). *The flow computed by a convex program of the form (SO) is optimal if and only if for every $m \in \mathcal{M}$ and $p_1, p_2 \in P^m$ with $x_{p_1} > 0$, we have $k'_{p_1}(x_{p_1}) \leqslant k'_{p_2}(x_{p_2})$.*

4.2.3 User equilibrium routing

By solving the SO problem we obtain the minimum average travel time for all the trips (commodities). On the other hand, some trips could have been disadvantaged with respect to others for the system's benefit. This causes the SO assignment to be unfair to some users.

A fair assignment is the User Equilibrium (UE). It induces a Nash equilibrium in which no user can improve its travel time (cost) by choosing a different route. It is also known as Wardrop equilibrium [6]. This equilibrium emerges when we consider the routing problem as a non-cooperative game, where each user, seen as a fraction of the flow, chooses the fastest route available according to the current traffic conditions. In this sense, users routed according to UE can be defined as self-interested or selfish.

Lemma 4.8 ([11]). *A feasible flow for instance $(G, \mathcal{M}, \mathbf{c})$ is at UE if and only if for every $m \in \mathcal{M}$ and $p_1, p_2 \in P^m$ with $x_{p_1} > 0$, we have $c_{p_1}(x_{p_1}) \leqslant c_{p_2}(x_{p_2})$.*

In particular, if a flow is at UE, all the paths of commodity m for which $c_p > 0$ have the same cost C_m.

Lemma 4.9 ([11]). *If a flow is at UE for instance $(G, \mathcal{M}, \mathbf{c})$, then the overall average cost (travel time) is*

$$\sum_{p \in P} x_p c_p(x_p) = \sum_{m \in \mathcal{M}} d^m C_m \tag{4.16}$$

This means that all users on the same trip will arrive at the same time to their destination under the user assignment.

Lemma 4.10 ([11]). *Given an instance $(G, \mathcal{M}, \mathbf{c})$, if there exists a feasible flow assignment, there always exists a feasible flow assignment at UE. Furthermore, given two assignments at UE, they will have the same total cost (travel time) (shown in Eq. 4.16).*

Lastly, let us define the UE routing problem as follows

$$\min_{\mathbf{x}} \sum_{e \in \mathcal{E}} \int_0^{x_e} c_e(t) dt \tag{4.17a}$$

s.t.

$$x_p \geqslant 0 \quad \forall p \in P \tag{4.17b}$$

$$\sum_{p \in P^m} x_p = d^m \quad \forall m \in \mathcal{M} \tag{4.17c}$$

$$x_e = \sum_{p \in P : e \in p} x_p \quad \forall e \in \mathcal{E} \tag{4.17d}$$

Where the constraints are the same as in the SO case, with the addition of 4.15e.

4.2.4 Differences between system optimal and user equilibrium routing

The SO and UE problems are very different. In particular, when considering the UE formulation, we trade-off total travel time for fairness and self-interest.

However, by looking at Lemmas 4.7 and 4.8 we can notice some similarities. The striking similarity between the characterizations of optimal solutions to the SO and UE problems was noticed early on [58], and provides an interpretation of an optimal flow as a flow at UE with respect to a different set of edge cost functions [11]. To make this relationship precise, denote the marginal cost of increasing flow on edge e by $c_e^*(x) = k_e'(x) = (x_e c_e(x))' = c_e(x) + x_e c_e'(x)$. Lemmas 4.7 and 4.8 yield the following corollary.

Corollary 4.11 ([11])**.** *Let $(G, \mathcal{M}, \mathbf{c})$ be an instance and $\mathbf{c}^*$ be the vector of marginal cost functions defined as above. Then, a flow feasible for $(G, \mathcal{M}, \mathbf{c})$ is optimal if and only if it is at UE for the instance $(G, \mathcal{M}, \mathbf{c}^*)$.*

4.2.5 Properties of path additions

Given a finite set of commodity path graphs $\mathcal{H}$ with flow-dependent edge costs c_e, we define function $\Lambda_O : 2^{\mathcal{H}} \mapsto \mathbb{R}$ which takes as input a subset of $\mathcal{H}$ and adds it to a fixed trip spanning tree defined over a fixed set of commodities and outputs the optimal value of the SO assignment and function $\Lambda_U : 2^{\mathcal{H}} \mapsto \mathbb{R}$ which does the same thing for UE.

Let us take as an example a set of commodity path graphs $\mathcal{A} \subseteq \mathcal{H}$; Λ_O and Λ_U are then computed as

$$\Lambda_O(\mathcal{A}) = \sum_{p \in P_{\mathcal{A} \oplus \mathcal{I}}} x_p^O c_p(x_p^O) \tag{4.18a}$$

$$\Lambda_U(\mathcal{A}) = \sum_{p \in P_{\mathcal{A} \oplus \mathcal{I}}} x_p^U c_p(x_p^U) \tag{4.18b}$$

Where $\mathbf{x}^O$ and $\mathbf{x}^U$ are respectively the solutions to the SO (4.15) and UE (4.17) flow problems.

Theorem 4.12. *Given two subsets of commodity path graphs $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{H}$, we have that $\Lambda_O(\mathcal{B}) \leqslant \Lambda_O(\mathcal{A})$, thus Λ_O is a monotonic non-increasing function of the set $\mathcal{H}$.*

Proof. The proof of Theorem 4.1 can be extended to this case. ■

Note that Braess' paradox represents a proof that the same theorem does not hold for Λ_U.

Now, we proceed to prove that Λ_O and Λ_U are not supermodular functions of set $\mathcal{H}' \subseteq \mathcal{H}$, defined as $\mathcal{H}' = \{x | x \in \mathcal{H} \wedge \forall y \in \mathcal{H},\ y \neq x \rightarrow \mathcal{E}_x \cap \mathcal{E}_y \cap \mathcal{E}_{\mathcal{I}} = \varnothing\}$, the set of commodity path graphs that do not have any common edges.

Theorem 4.13. *Λ_O and Λ_U are **not** supermodular set functions of $\mathcal{H}'$, and thus of $\mathcal{H}$.*

Proof. The counterexamples of Figure 4.3 and Figure 4.4 can be extended for this case. Here, for completeness, we show it in a more compact network, depicted in Figure 4.6.

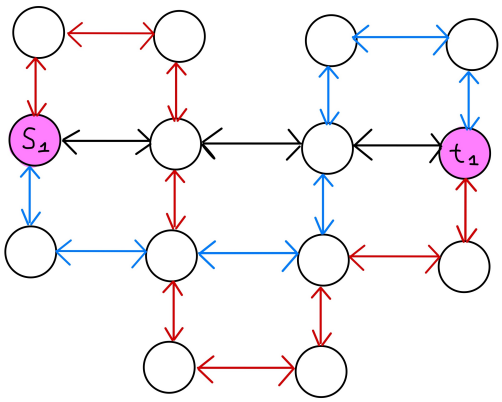


Figure 4.6: Compact counterexample of supermodularity when added commodity path graphs are allowed to have common nodes but not common edges.

$G_{\mathcal{I}}$ is shown in black, $G_{\mathcal{Y}}$ is shown in blue, and G_x is shown in red. We still consider $\mathcal{A} = \varnothing$, $\mathcal{B} = \mathcal{Y} = G_{\mathcal{Y}}$ and $x = G_x$, thus, Equation 4.8 still holds. This example is characterised by the following:

- There is only one trip $\mathcal{M} = \{(s^1, t^1, 5)\}$ with a demand of 5
- Every edge has a capacity $u_e = 10$
- Edges have a length $l_e = 1$, a speed limit $v_e^{max} = 1$

To compute the SO and UE assignments, we build this network in our macroscopic traffic simulator and we use the Frank-Wolfe algorithm (Sec. 5.1.2) to obtain the travel times. We confirm the obtained values by exact hand calculations. When solving Equation 4.9 for Λ_O we now obtain:

$$\begin{gathered} \Lambda_O(\varnothing) - \Lambda_O(\{x\}) \geqslant \Lambda_O(\mathcal{Y}) - \Lambda_O(\mathcal{Y} \cup \{x\}) \\ 30 - 29.28 \geqslant 27.47 - 24.97 \end{gathered} \tag{4.19}$$

While for Λ_U we obtain:

$$\begin{gathered} \Lambda_U(\varnothing) - \Lambda_U(\{x\}) \geqslant \Lambda_U(\mathcal{Y}) - \Lambda_U(\mathcal{Y} \cup \{x\}) \\ 30 - 30 \geqslant 30 - 26.66 \end{gathered} \tag{4.20}$$

Both equations do not hold. ■

4.2.6 Identical parallel paths

Let us move our attention to the case of a single commodity (s, t, d) and a set of parallel and identical commodity path graphs. In particular, we consider a set $\mathcal{H}'' \subseteq \mathcal{H}'$ in which

every path is parallel and connects s with t. All the paths have the same length l, the same maximum velocity v_{max} and the same capacity u.

Thanks to these assumptions, we can use the following flow assignment

$$x_p = \frac{d}{n} \quad \forall p \in P \tag{4.21}$$

Where $n = |P|$ is the number of parallel paths available. We can rewrite the objective function 4.15a of SO as

$$\min_{\mathbf{x}} \sum_{p \in P} x_p c_p(x_p) = d c_p \left(\frac{d}{|P|} \right) \tag{4.22}$$

Lemma 4.14. *Given an instance $(G_{\mathcal{A}\oplus\mathcal{I}}, \mathcal{M}, \mathbf{c})$ where $\mathcal{A} \subseteq \mathcal{H}''$ and $\mathcal{M} = \{(s,t,d)\}$, assignment 4.21 is an optimal solution for the UE problem*

Proof. By Lemma 4.8 we know that if we can show that $\forall m \in \mathcal{M}$ and $p_1, p_2 \in P^m_{\mathcal{A}\oplus\mathcal{I}}$ with $x_{p_1} > 0$, we have $c_{p_1}(x_{p_1}) \leqslant c_{p_2}(x_{p_2})$ then the assignment is at UE.

Given that all paths are identical and, under assignment 4.21, all flows are positive, we can rewrite

$$c_{p_1}(x_{p_1}) \leqslant c_{p_2}(x_{p_2}) \tag{4.23}$$

as

$$\frac{l}{v_{max}\left(1 - \frac{d}{|P_{\mathcal{A}\oplus\mathcal{I}}|u}\right)} \leqslant \frac{l}{v_{max}\left(1 - \frac{d}{|P_{\mathcal{A}\oplus\mathcal{I}}|u}\right)} \tag{4.24}$$

Which is always true. ■

Lemma 4.15. *Given an instance $(G_{\mathcal{A}\oplus\mathcal{I}}, \mathcal{M}, \mathbf{c})$ where $\mathcal{A} \subseteq \mathcal{H}''$ and $\mathcal{M} = \{(s,t,d)\}$, assignment 4.21 is an optimal solution for the SO problem*

Proof. By Lemma 4.7 we know that if we can show that $\forall m \in \mathcal{M}$ and $p_1, p_2 \in P^m_{\mathcal{A}\oplus\mathcal{I}}$ with $x_{p_1} > 0$, we have $k'_{p_1}(x_{p_1}) \leqslant k'_{p_2}(x_{p_2})$ than the assignment is at UE.

Given that all paths are identical and, under assignment 4.21, all flows are positive, we can rewrite

$$k'_{p_1}(x_{p_1}) \leqslant k'_{p_2}(x_{p_2}) \tag{4.25}$$

as

$$\left(\frac{d}{|P_{\mathcal{A}\oplus\mathcal{I}}|} \cdot \frac{l}{v_{max}\left(1 - \frac{d}{|P_{\mathcal{A}\oplus\mathcal{I}}|u}\right)} \right)' \leqslant \left(\frac{d}{|P_{\mathcal{A}\oplus\mathcal{I}}|} \cdot \frac{l}{v_{max}\left(1 - \frac{d}{|P_{\mathcal{A}\oplus\mathcal{I}}|u}\right)} \right)' \tag{4.26}$$

Which is always true. ■

Theorem 4.16. *Λ_O and Λ_U are supermodular functions of the set $\mathcal{H}''$*

Proof. For brevity, the proof can be found in Appendix A.4. ■

4.3 Summary

In this chapter, we have shown that the rather intuitive property of supermodularity of path additions to transport networks where AVs are routed does not hold in the general case of multiple AVs' trips and in the simpler case of a single trip. We have proven this for the min-cost multi-commodity flow problem and for the SO and UE routing formulations.

We have also shown that in the more specific case of parallel paths, we have diminishing returns upon the addition of new paths. We have proven this in the case of parallel paths with different costs for the minimum-cost network flow problem and in the case of identical parallel paths in the flow-dependant cost formulation, where we have also shown that the solutions to the SO and UE routing problems coincide.

Chapter 5

Transport network design

In this chapter, we introduce the problem of optimising the transportation network for Autonomous Vehicles (AVs) routed according to the User Equilibrium (UE), which is a self-interested and fair routing paradigm. We formalise two different optimisation tasks.

In the first one, we consider a *trip spanning tree* (Sec.4.1.2), resulting from a set of AVs trips, to which we aim to add *commodity path graphs* in order to minimise the total system travel time. In other words, we start with a minimal transportation network that barely permits agent movement and we seek optimal ways to extend it. To solve this problem, we propose a greedy algorithm.

In the second one, we start with a network topology (e.g. a city), where we believe that Braess' paradox occurs, and we seek to reduce the edge capacities in order to make the selfish users behave optimally (minimise the total system travel time). Our capacity reduction is similar to a road pricing scheme, in the sense that we show some roads to users as more costly than they really are, but costs are not fictitious as they are always mapped to edge capacities, which are real topology parameters. To solve this problem, we employ a Genetic Algorithm (GA) and Reinforcement Learning (RL).

The chapter is structured as follows: In Section 5.1 we introduce the bilevel formulation of network design, in Section 5.2 we discuss network design through path additions, in Section 5.3 we discuss network design using capacity reductions, and, finally, Section 5.4 summarises the chapter.

5.1 The bilevel problem

The network design problem (NDP) is formulated as a bilevel programming optimisation task. Its structure is depicted in Figure 5.1.

At the upper level, the network designer is interested in modifying the network topology in order to minimise the total system travel time. At the lower level, users receive a transportation network and route greedily in it according to the UE. Note that the network designer cannot control the users' routing directly, but only influence it through topology modifications.

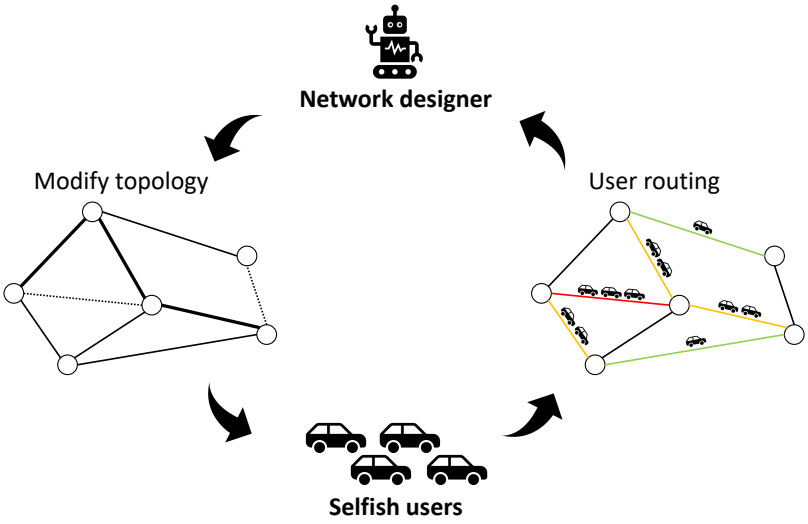


Figure 5.1: Network design as a bilevel optimisation problem. The designer can modify the network topology and observe how user redistribute.

The problem has the following structure:

$$\min_{\mathbf{y}} F(\mathbf{y}, \mathbf{x}(\mathbf{y})) \tag{5.1a}$$

s.t.

$$\mathbf{G}(\mathbf{y}, \mathbf{x}(\mathbf{y})) \leqslant \mathbf{0} \tag{5.1b}$$

where $\mathbf{x}(\mathbf{y})$ is determined as the result of:

$$\min_{\mathbf{x}} f(\mathbf{y}, \mathbf{x}) \tag{5.2a}$$

s.t.

$$\mathbf{g}(\mathbf{y}, \mathbf{x}) \leqslant \mathbf{0} \tag{5.2b}$$

F is the objective function of the network designer, $\mathbf{y}$ is the topology decision vector, $\mathbf{G}$ is the constraint set of the upper level decision vector, f is the objective function of the users, $\mathbf{x}$ is the decision vector of the users, and $\mathbf{g}$ is the constraint set of the lower level decision vector.

Even if the upper and lower level are both convex optimisation problems, the bilevel problem in its entirety is non-convex [60], as users' decisions act as a nonlinear constraint for the upper level. Thus, traditional convex optimisation solution methods are not applicable to this problem.

5.1.1 Upper level

In this chapter, we will see two different formulations of the upper level problem: path additions and capacity reductions. They are discussed in Section 5.2 and Section 5.3 respectively.

5.1.2 Lower level

The lower level is always fixed: Given a network and a set of trips with their demands, also called Origin-Destination (OD) matrix, compute the UE routing flows. In other words, solve Problem 4.17 to obtain $\mathbf{x}^U$. We tackle it using two approaches.

For the first task (5.2) we employ microscopic traffic simulations using SUMO [2]. To compute the UE traffic assignment, we reproduce the work from self-aware routing [3], without considering the probabilistic component. We also use SUMO's implementation of dynamic UE routing [15] and a one-shot assignment approach[1].

For the second task (5.3) we employ traditional macroscopic traffic simulation. We build our own simulator which is compatible with network and trips definitions in the `tntp` format[2]. We implement the Method of Successive Averages [61] and the Frank-Wolfe convex optimisation algorithm [4] that can both efficiently compute the UE and SO traffic assignments. Our traffic assignment algorithm has been tested on all the networks available in this *traffic research repository*[2] and has reported correct results. We make the simulator and assignment solver available as standalone products. They are already being used by other researchers.

5.2 Path additions

We now formulate the path additions network design problem. This problem considers a trip spanning tree and a fixed OD matrix to be given. A set of possible commodity path graphs $\mathcal{H}$ is also defined. All these graphs are subgraphs of a graph template $G_{\mathcal{T}}$ which defines the topology to follow during construction. The set $\mathcal{H}$ of possible additions is constructed by considering the k-shortest paths between each trip's source and its destination in $G_{\mathcal{T}}$. Therefore $|\mathcal{H}| = N_T k$, where N_T is the number of trips and k is the number of possible additions considered for each trip. We are interested in finding the optimal subset $\mathcal{A} \subseteq \mathcal{H}$ in order to minimise the total system travel time when $\mathcal{A}$ is subject to cardinality constraints. For our purpose, the set of path graph additions $\mathcal{A}$ is equally identified by the graph $G_{\mathcal{A}} = \cup_{G \in \mathcal{A}} G$. The problem is formalised as follows.

$$\underset{\mathcal{A} \subseteq \mathcal{H}}{\operatorname{argmin}} \Lambda_U(\mathcal{A}) \tag{5.3a}$$

$$\text{s.t.}$$

$$|\mathcal{A}| \leqslant N_p \tag{5.3b}$$

$$|\mathcal{V}_{\mathcal{A}} \setminus \mathcal{V}_{\mathcal{I}}| \leqslant N_v \tag{5.3c}$$

$$|\mathcal{E}_{\mathcal{A}} \setminus \mathcal{E}_{\mathcal{I}}| \leqslant N_e \tag{5.3d}$$

Where $G_{\mathcal{A}} = (\mathcal{V}_{\mathcal{A}}, \mathcal{E}_{\mathcal{A}})$ and N_p, N_v, N_e are cardinality constraints on the number of added paths, nodes, and edges respectively. Recall the definition $\Lambda_U(\mathcal{A}) = \sum_{p \in P_{\mathcal{A} \oplus \mathcal{I}}} x_p^U c_p(x_p^U)$, which computes the total travel time of users following the UE.

[1] `https://sumo.dlr.de/docs/Tools/Assign.html#one-shotpy`
[2] `www.github.com/bstabler/TransportationNetworks`

Clearly, problem 5.3 is the upper level of our general bilevel network design formulation. Here, the topology decision variables $\mathbf{y}$ consist in a 0/1 associated to each element in $\mathcal{H}$, indicating if it is included in $\mathcal{A}$ or not. In particular, in addition to the non-convexity resulting from the bilevel structure, the discrete decision variables of Problem 5.3 make it an NP-Hard discrete network design problem [47]. Thus, efficient exact solutions are computationally infeasible and heuristics and approximations can be used. In this work, we use a greedy algorithm that, at each design iteration, adds to the current set $\mathcal{A}$ the path that reduces the total travel time the most and does not break the constraints. Having proven the non-supermodularity of Λ_U, we cannot provide sub-optimality bounds for our algorithm.

Problem 5.3 is relevant when the template of the transport network is available (e.g. a warehouse) and the network designers want to be informed on how different possible layouts will affect the overall routing performance of AVs. Thus, they can formulate the problem through path additions and restructure the network topology online when the AVs origin, destinations and demands change. Our method, in fact, takes as input a network template and AVs trip requests and, in a trivial time, outputs an optimal sub-topology of the original layout which optimises the total travel time and is Braess' paradox free.

5.3 Capacity reduction

The network design problem can also be tackled starting from an existing graph. In this problem, we are given a graph $G = (\mathcal{V}, \mathcal{E})$ and a fixed OD matrix. The goal is to decrease the edge capacities seen by the users in order to minimise the total travel time. The problem is formulated as follows.

$$\min_{\mathbf{y}} \sum_{e \in \mathcal{E}} x_e^U(\mathbf{y}) c_e(u_e, x_e^U(\mathbf{y})) \tag{5.4a}$$

s.t.

$$0 \leqslant y_e \leqslant u_e \quad \forall e \in \mathcal{E} \tag{5.4b}$$

$$y_e \in \mathbb{R} \quad \forall e \in \mathcal{E} \tag{5.4c}$$

Where the designer decision vector $\mathbf{y} = (y_0, y_1, \ldots, y_e, \ldots, y_N)$ is the vector of the users' edge capacities with $N = |\mathcal{E}|$, which cannot exceed the real capacities $\mathbf{u}$. $\mathbf{x}^U(\mathbf{y})$ is the solution to the lower level problem in which the users distribute according to the UE using $\mathbf{y}$ as edge capacities instead of $\mathbf{u}$. Note that the reduced capacities $\mathbf{y}$ are only seen by users. In the upper level (Problem 5.4) we still use the original capacities $\mathbf{u}$ to compute the total travel time. This can be seen as implementing tolls for the users just by showing some roads as having less capacity than they really have. The cost $c_e(y, x)$ is the BPR function 4.14, computed as:

$$c_e(y, x) = c_e^0 \left(1 + 0.15 \left(\frac{x}{y}\right)^4\right) \tag{5.5}$$

Where y is the capacity, x is the flow, and c_e^0 is the edge free-flow travel time. Note that any travel time link function could be used.

This problem may seem very similar to traditional network design problems [47], but it has some key differences. Firstly, we leverage the insight that, due to the Braess' paradox, the current network may be used inefficiently; we aim to solve this issue by allowing continuous capacity decreases (note that the case of entire edge removals is contained in our problem and can be achieved when $y_e = 0$), while traditional network design only considers continuous capacity increases under a construction budget. This is a radical perspective, as our method, unlike continuous network design, is able to optimise networks without the need to physically modify them. Secondly, it brings together the concepts of road pricing policies and environment design by increasing the cost of non-optimal roads through capacity reductions. The main innovative insight is that we are able to optimise networks used by selfish users just by simulating the removal of infrastructure. Lastly, our method optimises the system's throughput while having users routing according to UE at all stages, thus achieving benefits for the system while avoiding the unfairness of SO routing.

5.3.1 Reinforcement learning

We formulate the capacity reduction network design problem from a RL perspective. Under this view, we consider a single RL agent as the network designer, which can take actions to move in the search space of problem 5.4. The agent can decide to modify the capacity of one edge per time step and observe the users' routing response following such modification. This characterises its reward.

Environment

The agent has full observability of the state space. The state at time t is a vector $\mathbf{s}_t \in \mathbb{R}^{N \times F}$ where N is the number of edges in the transport network and F is the number of features considered for each edge. In this work we consider only two features: current flow x on the edge and current visible edge capacity y. Note that by having access to these features, the agent can derive the OD matrix. The transport network topology is given to the agent during initialisation. The agent has two actions for each edge, $a_t \in \{0, 1, \dots, 2N-1\}$, which lead respectively to decreasing and increasing the visible capacity y_e of the edge as:

$$\forall e \in \mathcal{E}, \quad y_e = \begin{cases} \max(0, y_e - \Delta_u u_e), & \text{if } a_t = 2e \\ \min(u_e, y_e + \Delta_u u_e), & \text{if } a_t = 2e+1 \\ y_e, & \text{otherwise} \end{cases} \tag{5.6}$$

With $0 \leqslant \Delta_u \leqslant 1$. In this work we use $\Delta_u = 0.1$ (actions move y_e with steps of 10% of the real capacity). For example, if at the beginning of an episode we have $\mathbf{y} = \mathbf{u}$ and the agent selects ten times in a row action 2, he will bring the visible capacity y_1 of edge 1 to 0, by then selecting action 3 ten times in a row, the agent goes back to the initial state. The reward $r_{t+1} \in \mathbb{R}$ is defined as the percentage of relative improvement in the

total travel time after action a_t.

$$r_{t+1} = 100 \left(1 - \frac{\text{Total travel time after } a_t}{\text{Total travel time before } a_t}\right) \tag{5.7}$$

Graph neural network model

In order to train the agent we use PPO (Sec. 2.5) and the RLlib [62] framework. For both the actor and the critic we use a Graph Neural Network (GNN). The idea behind using GNNs in our model is to leverage the intrinsic graph structure of the road network during the learning process, enabling sharing information between neighbouring roads using Graph Convolutions [63]. The GNNs in our model are line graphs of the transportation network, thus, every node in the GNN represents an edge in the road network.

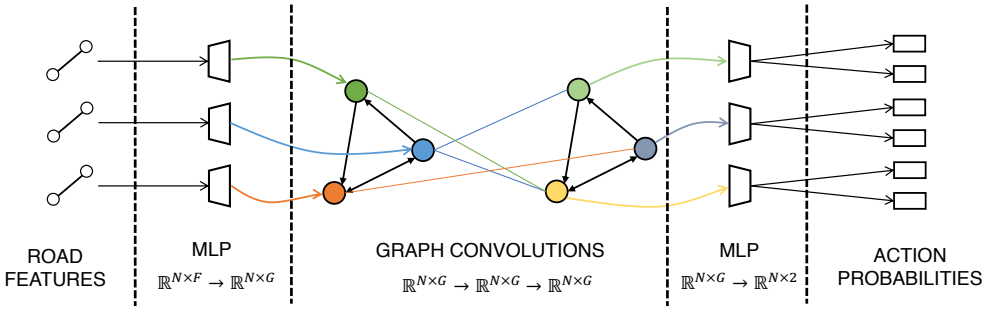
The models are depicted in Figure 5.2. Although the actor and the critic do not share any parameters, they share the same architecture for the first part of the pipeline. The model input is the state of dimension $\mathbb{R}^{N \times F}$ where N is the number of roads and $F = 2$ are the road features. The state vector is passed through a Multi Layer Perceptron (MLP) which expands the feature dimension $\mathbb{R}^{N \times F} \to \mathbb{R}^{N \times G}$. Each of the N road vectors is then associated to the corresponding node in the GNN, which has two graph convolution layers [63] that preserve the dimension $\mathbb{R}^{N \times G}$. The architecture of the actor and the critic differs from this point on. For the actor, the G features of each GNN node go through an MLP which maps them to the probabilities of the two edge actions $R^{N \times G} \to \mathbb{R}^{N \times 2}$. For the critic, we employ global mean pooling of the GNN node vectors $R^{N \times G} \to \mathbb{R}^{1 \times G}$ and finally pass it through an MLP which maps it to the state value $R^{1 \times G} \to \mathbb{R}^{1 \times 1}$. To all the layers we apply hyperbolic tangent non-linear activations and, for the actor head, we use a softmax activation to get the action probabilities from the logits. We use an embedding dimension $G = 64$.

5.3.2 Genetic algorithm

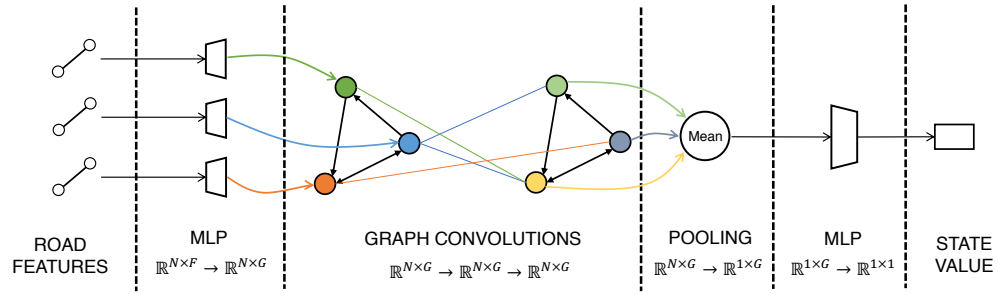
The Genetic Algorithm (GA) used to solve the capacity reduction problem follows the outline introduced in Section 2.4. Here, an individual is represented by the vector $\mathbf{z} = (z_0, z_1, \ldots, z_N)$ such that, for an edge e, the visible capacity $y_e = z_e u_e$. Therefore, the elements of $\mathbf{z}$ are all real numbers between 0 and 1 included, representing the portion of real capacity shown to the users. The elements of the vector $\mathbf{z}$ are also referred to as the genes of the individual.

During **phase 0**, we initialise a population of n_{POP} individuals. All of the individuals start with all the genes set to 1. This has shown to be better in practice than initialising genes randomly, because the optimal solution will probably be located in the vicinity of the initial transport network structure[3]. A GA iteration, also referred to as a generation, is structured as follows. During **phase 1**, we compute the fitness of each individual as the total travel time that the users would experience in the network characterised by its $\mathbf{z}$. The fittest individual is picked and compared with the previous fittest in order to update the best solution obtained so far. Then, the n_{CARRY} fittest individuals are directly added to the next generation and skip the following phases. To fill the remaining

[3]We are removing infrastructure, random initialisation leads to a very non-optimal start.



(a) Actor model



(b) Critic model

Figure 5.2: PPO actor and critic models.

$n_{\text{POP}} - n_{\text{CARRY}}$ spots for the next generation, we proceed as follows: For each spot to fill, we randomly choose $\frac{n_{\text{POP}}}{3}$ individuals from the current generation and select the fittest, that is then placed in that spot. During **phase 2**, these $n_{\text{POP}} - n_{\text{CARRY}}$ individuals are grouped into pairs. For each pair, we perform crossover at a random index in the pair's gene vectors according to a probability p_{CROSS}. If crossover is performed, each pair is substituted by the respective children pair. Finally, during **phase 3**, with a probability p_{MUT}, each gene of the $n_{\text{POP}} - n_{\text{CARRY}}$ individuals is mutated according to

$$z_e = \max(0, \min(1, z_e + \mathcal{U}(-\Delta_u, \Delta_u))) \tag{5.8}$$

Where $\mathcal{U}(-\Delta_u, \Delta_u)$ indicates a value sampled from the uniform distribution between $-\Delta_u$ and Δ_u. The step $\Delta_u = 0.1$, as in the RL actions, helps clipping the maximum mutation by giving a boundary on the difference between the new and old gene. This component is somewhat similar to the concept of clipping in the PPO loss function. After mutation, the next generation restarts from phase 1. As we do not initialise the population randomly, the main power of our evolutionary formulation resides in the design of the mutation function..

5.4 Summary

In this chapter, we have formulated two innovative network design tasks. We tackled the problem of designing networks for selfish users routed with UE first from a bottom-up and then from a top-down perspective. We have seen how both these optimisation tasks can be of interest for network designers and can help tackle real-world problems. In particular,

unlike previous work, we are not interested in efficiently building new infrastructure under construction costs, but we focus on bridging the gap between SO and UE by making better use of the current transport network. A wide range of solution approaches are proposed, such as a greedy algorithm and a genetic algorithm. We also formulate the problem under a RL perspective, where a network designing agent has to traverse a non-convex optimisation space leveraging the power and structural expressiveness of GNNs. In the next chapter, we present an evaluation of the proposed solutions.

Chapter 6

Evaluation

In this chapter, we evaluate the proposed solutions for the two network design tasks formulated. We also compare different routing paradigms and benchmark our implementation of Self-aware routing [3] in a range of urban scenarios in SUMO. Experimental evidence of Theorem 4.13 is provided. Path additions are evaluated in custom grid networks, while capacity reductions are run on the Braess' network as well as on six real-world transport networks: Anaheim (USA), Barcelona (Spain), Chicago (USA), Eastern Massachusetts (USA), Sioux Falls (USA), and Winnipeg (Canada).

6.1 Path additions

In this section, we provide evaluation results for the path additions network design problem (Equation 5.3).

6.1.1 Experimental setup

Generating networks

The experiments for this problem are carried out in the microscopic traffic simulator SUMO [2]. We focus on studying the problem of path additions for vehicle routing in the context of urban transport networks. For this reason, our graph templates $G_{\mathcal{T}}$ are lattice grids, resembling the topology of cities. These networks represent a portion of a city or a district. Grids have been chosen since ancient Roman times as the main topology for urban environments and, still today, characterise the outline of many major cities in the world (e.g., New York). An example of a 5×5 grid template is shown in Figure 6.1.

To allow custom grid network generation, we implement a framework that seamlessly integrates the NetworkX[1] python library and SUMO networks. Users can build custom grid networks and specify a wide range of network parameters. In Table 6.1 we report the main parameters as well as the default values we use in our simulations.

[1] `www.networkx.org`

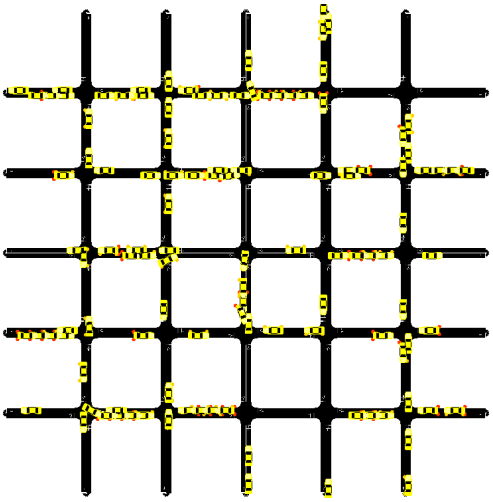


Figure 6.1: Custom generated 5×5 grid network in SUMO.

Table 6.1: Transport network parameters used in SUMO.

Parameter name	Default
Grid width (N_G)	10
Grid height	N_G
Road length	100 m
Road length standard deviation	0
Road speed limit	13.9 m/s
Number of lanes	1

Generating trips

After the network is generated, it is possible to generate vehicles' trips and demands, also called origin-destination (OD) matrix. Our trip generation tool always generates trips that start and end from the outer nodes of the grid. Two different modes are available: "left to right" and "random". "Random" selects origins and destinations randomly, while "left to right" selects origins and destinations respectively from the left and right sides of the grid. In this work we use the "left to right" mode. Also in this case, the user can select from a wide range of parameters. In Table 6.1 we report the main parameters as well as the default values we use in our simulations.

Table 6.2: Vehicle parameters used in SUMO.

Parameter name	Default
Number of trips (N_T)	5
Number of vehicles per trip (N_V)	100
Vehicles' max acceleration	2.6 m/s^2
Vehicles' max deceleration	4.5 m/s^2
Vehicles' length	5 m
Minimum gap between vehicles	2.5 m
Speed standard deviation	0.2

N_G, N_T, and N_V are the parameters we will vary in our simulations to obtain results for different city sizes and congestion levels. All other parameters will be kept as default

in the experiments.

Routing algorithms

As already mentioned in Section 5.1.2, we implement various methods to route vehicles. All of them route vehicles according to the UE principle as it is the most fair and studied solution. The congestion function used to compute the costs is that of Greenshields (Eq. 4.13). Firstly, we reproduce from scratch the Self-aware algorithm proposed in [3]. In particular, we implement the pseudo-code at [3, p. 1524], eliminating the uncertainty components as they are not relevant to us. It is important to notice a major typo in the published paper at line 4 of the algorithm, where `argmax` should be replaced with `argmin`. We also use SUMO's implementation of dynamic UE (DUE) routing [15] and a one-shot assignment approach[2]. Lastly, users can also be routed according to Dijkstra's shortest-paths algorithm [64], which sends users on the shortest distance paths, ignoring congestion delays. All these routing algorithms are pluggable into our framework and constitute an useful tool for users willing to benchmark their solutions.

Optimisation setup

Once the network $G_{\mathcal{T}}$ has been created, the OD matrix has been generated, and the routing algorithm has been chosen, the optimisation pipeline can be set up. For each of the N_T trips generated, we compute the $(k+1)$-shortest paths between its origin and destination in $G_{\mathcal{T}}$ using Yen's algorithm [65]. Then, the *trip spanning tree* $G_{\mathcal{I}}$ is created by using the shortest path for each trip. The remaining k paths for each trip form the search space $\mathcal{H}$, with $|\mathcal{H}| = N_T k$. In the experiments we present, paths from $\mathcal{H}$ will be iteratively added to $G_{\mathcal{I}}$. After every addition, we reroute all the vehicles and observe the new total travel time. If after an addition the total travel time increases due to Braess' paradox, the users are forced to ignore the added path. This way we are able to guarantee the monotonicity of travel time with respect to path additions. For brevity, in our experiments we will vary the constraint N_p and leave N_v and N_e equal to infinity. A graphical illustration of how path additions look in SUMO is shown in Figure 6.2.

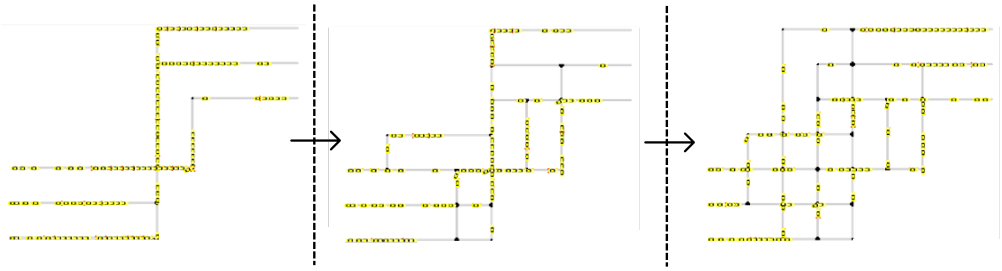


Figure 6.2: Illustration of path additions in SUMO.

6.1.2 Routing algorithms comparison

We benchmark our implementation of Self-aware routing [3] against the other state-of-the-art algorithms discussed in the experimental setup section. Results are shown in

[2] `https://sumo.dlr.de/docs/Tools/Assign.html#one-shotpy`

Figure 6.3. The origin of the axes represents the trip spanning tree ($N_p = 0$). Moving to the right on the x axis, paths are incrementally added. In this scenario, added paths are chosen randomly as we are not interested in evaluating the network optimisation, but we want to evaluate the different routing performance on networks created randomly. We evaluate two grid sizes: $N_G = 5, 10$ and two congestion scenarios: $N_T = 3, 5$, all with $N_V = 250$. We observe how Self-aware performs better[3] than DUE [15], which is currently the most accurate (and slowest) routing algorithm available with SUMO. In particular, Self-aware is substantially faster. In the four scenarios presented, DUE took on average 94.11s to perform a single routing assignment, while Self-aware took on average 8.78s. Thus, DUE showed to be 1072% slower than Self-aware.

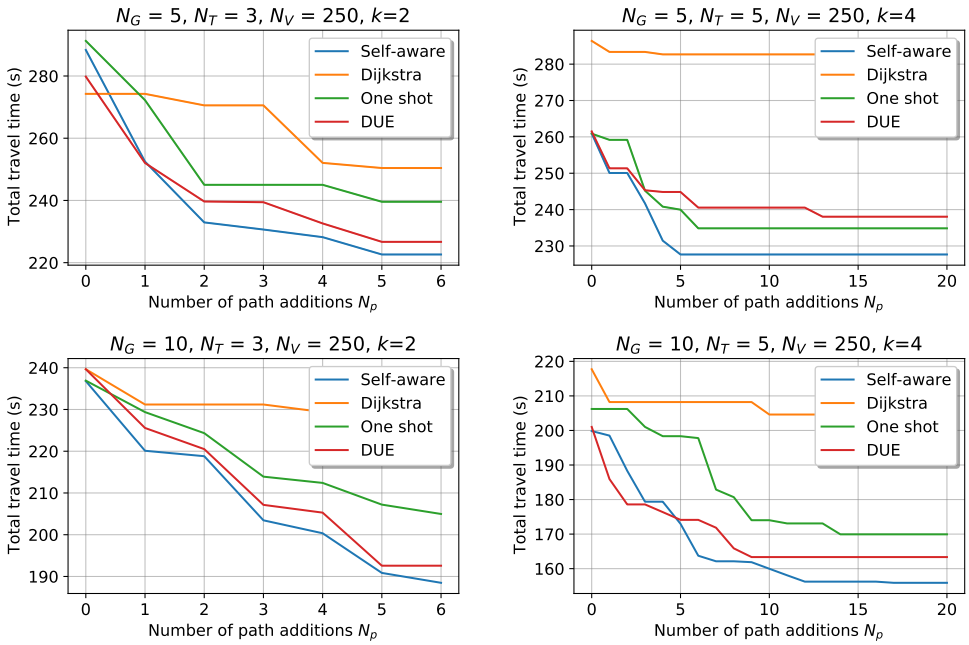


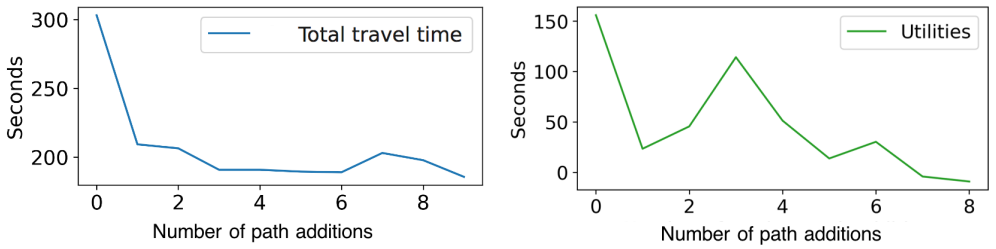
Figure 6.3: Total travel time comparison of routing algorithms under random incremental path additions in SUMO. The simulation parameters are shown on top of the plots.

6.1.3 Non-supermodularity

In this section, we briefly provide experimental confirmation of Braess' paradox and the absence of supermodularity in our framework. We consider a simple scenario with $N_G = 5, N_T = 1, N_V = 50$ and random path additions as in the previous case. The results are shown in Figure 6.4.

In Fig. 6.4a we see that, if the travel time is not forced to be monotonic, some edge additions can cause an increment in total travel time, as it is the case for addition number 7. In Fig. 6.4b, we plot a more subtle concept: utilities. We do this by taking a path p from our possible additions set and keeping it on a side. Then, the incremental addition process is performed by adding paths from the set $\mathcal{H} \setminus \{p\}$. After every addition, we compute the total travel time T on the resulting network and then add p to the network and recompute the total travel time T_p. The utility of the addition of p at

[3]In terms of total travel time.



(a) Braess' paradox

(b) Non-supermodularity

Figure 6.4: Experimental evidence of Braess' paradox and non-supermodularity in a simple scenario with $N_G = 5, N_T = 1, N_V = 50$ and random path additions

each step is $T - T_p$, this can also be called benefit or return. Following the property of supermodularity (Sec. 2.3), this utility should be diminishing (monotonic non-increasing) for all the possible choices of p. In this experiment we show that this is not the case, confirming experimentally the result of Sec. 4.2.5.

6.1.4 Greedy additions

In this section, we evaluate our solution to problem 5.3. We propose a greedy algorithm that builds the set $\mathcal{A}$ iteratively. At each step, the algorithm evaluates the travel time that would result after adding each of the remaining feasible[4] paths individually to the current network and chooses the one that results in the greatest decrease in total travel time. We compare this solution with an algorithm that, at each step, picks a random addition among the feasible[4] ones. For the random algorithm, we plot the mean and standard deviation of the results averaged between 10 different runs. The number of samples is limited to 10 because of the high computational cost of each simulation step. The routing algorithm we use is Self-aware, chosen based on the results from Section 6.1.2. Due to the fact that the problem formulation has been introduced by us, we were not able to use other existing solutions for further comparison.

In Figure 6.5 we can see the results. We evaluate three grid sizes: $N_G = 5, 10, 20$ which correspond to the rows of the figure. In the columns, we vary the number of trips: $N_T = 2, 5$. The number of vehicles per trip is always fixed to $N_V = 100$. These parameter sets have been chosen to represent typical demands for a central city district. On the x axis we can see the difference between the random and greedy performance obtained by varying the N_p constraint from 0 to $|\mathcal{H}|$.

We can observe how, in all scenarios, the first additions are the most effective, leading to the greatest decreases in travel time. However, this initial decrease is always followed by a flattening point, where additions start to consistently have very little impact. This fact highlights a very important intuition: *if infrastructure additions are chosen wisely, just a few impactful network extensions are sufficient to obtain a near optimal solution.* This is clearly shown in the top right plot of Figure 6.5, where the first 4 greedy additions are sufficient to match the performance of 10 random additions.

There is another important result, which may appear a little subtle. By forcing monotonicity of total travel time in our addition process, we counteract Braess' paradox.

[4]That satisfy the constraints.

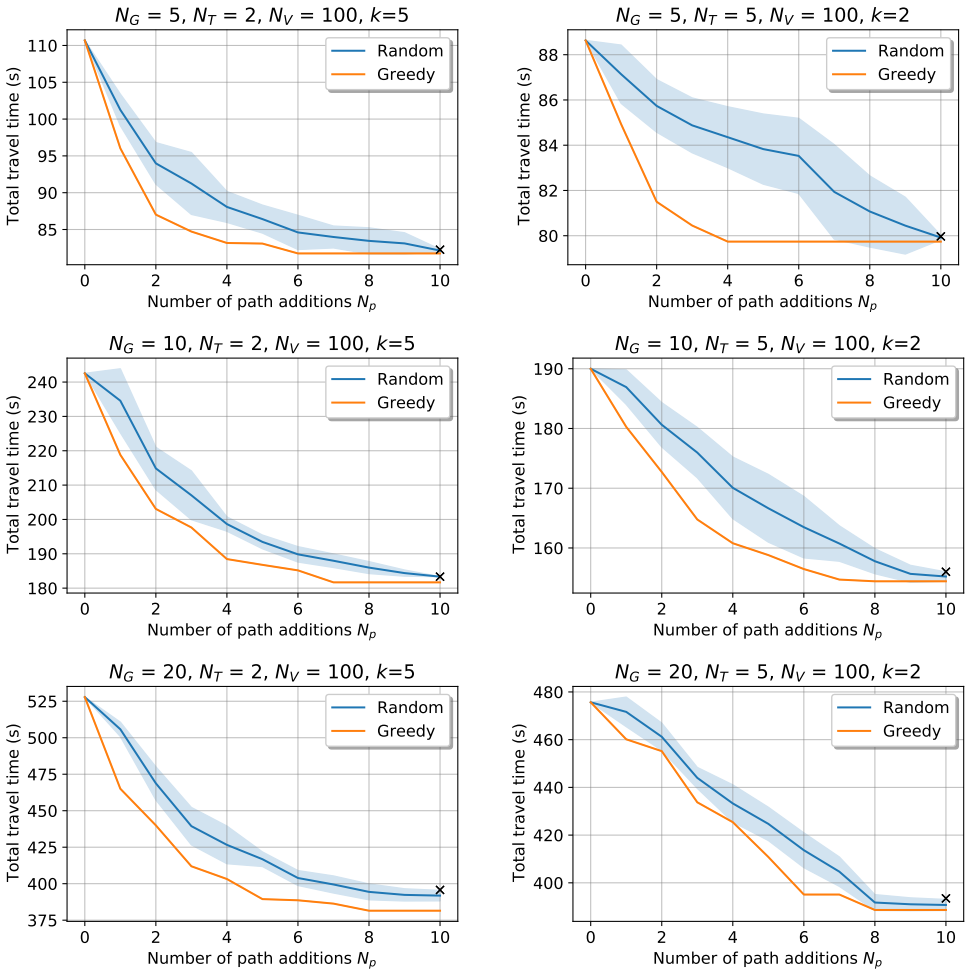


Figure 6.5: Total travel time comparison of greedy and random path additions in SUMO. The black $\times$ marker shows the total travel time of the network containing all the additions without going thorough the iterative process. Random results are averaged between 10 runs. The simulation parameters are shown on top of the plots.

Let's take as an example the bottom left plot of Figure 6.5. We can imagine that, during the greedy process, additions number 9 and 10 may have had no impact or, worse, negative impact. However, we do not see a spike like in Figure 6.4a, because we force the vehicles not to use the path just added if it leads to worse performance. This process has an important benefit: if we look at the networks when $N_p = 10$ (all the paths in $\mathcal{H}$ have been added) we can compute the total travel time, which is shown with a black $\times$ marker. This time differs from the total travel time of greedy, which got to this full network trough iterative additions. This is because, as we said, after every addition, the greedy algorithm ignores paradox-inducing paths. The distance between the black marker and the orange line represents the gained travel time. In Table 6.3 we show this benefit in terms of relative percentage of improvement in total travel time for the 6 scenarios of Figure 6.5. This data hints to a correlation between improvement and grid size, which could be interpreted as an increased likelihood of Braess' paradoxes in bigger networks.

Table 6.3: Percentage of improvement in total travel time obtained through iterative greedy additions instead of routing in the full network. The highest value is shown in bold.

	$N_T = 2$	$N_T = 5$
$N_G = 5$	0.62%	0.29%
$N_G = 10$	0.91%	1.03%
$N_G = 20$	**3.60%**	1.24%

6.2 Capacity reduction

In this section, we provide evaluation results for the capacity reduction network design problem (Equation 5.4). This problem formulation is highly innovative, thus, no existing solutions from the reviewed literature could be applied to it directly. For this reason, this section does not aim to compare our solutions with other approaches, but simply demonstrates empirically that significant total travel time improvements can be obtained only through capacity reductions in real-world cities and transport networks. In fact, comparison is ineffective even with a random baseline, as, reducing capacities randomly, always leads to catastrophic results.

6.2.1 Experimental setup

Transport networks used

For our evaluation, we use seven transport networks contained in this *traffic research repository*[5]. The repository contains detailed descriptions of each network. The networks are imported into our macroscopic traffic simulator. Congestion is modeled using the BPR delay function (Eq. 5.5). The networks are described in Table 6.4.

Table 6.4: Transport networks imported in our macroscopic traffic simulator.

Transport network	Nodes	Edges	Trips
Braess [1]	4	5	1
Anaheim (USA)	416	914	1406
Barcelona (Spain)	930	2522	7922
Chicago (USA)	933	2950	142890
Eastern Massachusetts (USA)	74	258	5476
Sioux Falls (USA)	24	76	576
Winnipeg (Canada)	1040	2836	4345

Genetic algorithm setup

The parameters used for the GA have been chosen by hand tuning. In the experiments we set $n_{\text{POP}} = 10$, $n_{\text{CARRY}} = 2$, $p_{\text{CROSS}} = 0.9$, $p_{\text{MUT}} = 0.3$. The population is kept small as, on big networks, solving the routing equilibrium for each individual becomes computationally demanding.

[5] `www.github.com/bstabler/TransportationNetworks`

Apart from the mutation function introduced in Section 5.3.2, which we call *continuous mutations*, we introduce two other mutation formulations. The first one looks at modifying the capacities between 0 and 1 in the following way:

$$z_e = \begin{cases} 1, & \text{if } z_e = 0 \\ 0, & \text{if } z_e = 1 \end{cases} \tag{6.1}$$

We call this formulation *discrete removals* (Disc Rem) as it aligns with the Discrete Network Design Problem [47], which looks at removing entire roads. The second one aligns with the RL agent actions and allows capacities to mutate with discrete Δ_u steps. It is defined as:

$$z_e = \max(0, \min(1, z_e + \Delta_u(2\text{Bern}(0.5) - 1))) \tag{6.2}$$

Where Bern(0.5) indicates a Bernoulli random variable that assumes value 0 or 1 with probability 0.5. We call this formulation *discrete mutations* (Disc Mut) as the capacities can only mutate of discrete Δ_u steps.

Reinforcement learning setup

The parameters used for the RL agent have also been chosen by hand tuning. They are shown in Table 6.5. The episodes are stopped after 200 agent interactions or when the agent degrades the original performance by more than 1%. The episode is also stopped if the agent is able to obtain SO performance.

Table 6.5: Reinforcement learning parameters.

Training		PPO		Model	
Batch	6000	ϵ	0.2	G	64
Minibatch	200	γ	0.995	Activation	Tanh
SDG Iterations	20	λ	1.0		
Workers	16	β	0.2		
Learning rate	5e-5	c_2	0.001		

Optimisation setup

The optimisation always starts from the full network, with $\mathbf{y} = \mathbf{u}$. On this full network, we calculate the total travel time of the users for both UE and SO routing. We denote the UE total travel time as the *original* time, as it is our starting point. We denote the SO total travel time as the *optimal* time. Note that *optimal* does not refer to the travel time of the optimal solution to our task, but it constitutes a lower bound[6] to that value. This is because we are trying to push the UE time of a capacity decreased network towards the SO time of the original network, but we do not have guarantees that this value can be reached, so the *optimal* is a lower bound on the time of the optimal solution.

To evaluate our solutions, we use the *absolute improvement* metric. Which is calculated as the percentage of improvement in total travel time form the *original* time.

[6]Lower bound in terms of travel time, the optimal solution to the task can thus be at most as good as the *optimal* value.

6.2.2 Total travel time improvement

Braess' network

To evaluate the correctness of our solutions, we test them on the Braess' network, shown in Figure 2.2b. For this network, we know that the optimal solution consists in removing the central link. In our case, this corresponds to setting its capacity to 0 ($y_e = 0$).

After 10 training iterations, the RL agent is able to learn the optimal policy, which consists in reducing 10 times in a row the capacity of the central edge, until it becomes 0. The same is valid for all the GA formulations, which converge to the optimal in less that 5 iterations.

Real-world networks

We then move to test on the six real-world networks.

Here, we observe that the RL approach fails to generalise and achieves near original performance. The idea for the RL agent is to train on different trip demands in order to be able to generalise to new demands for a given network. While on a small artificial example, such as Braess' network, the agent is able to achieve this task, on real-world networks, it struggles. We have identified two possible causes for this result. Firstly, Braess' paradox is dependent on the demands. As we can see in the example by Braess' (Sec. 2.2.2), the paradox happens with a demand of 6, and does not happen with lower demands. This informs us that making the agent generalise on random demands could mean to generalise to highly different paradox scenarios. Secondly, the state space of the agent is actually the search space of a non-convex optimisation task, that grows proportionally to the network size. Therefore, asking the agent to traverse this huge space without guidance from an expert (which is common practice, but unfeasible in our case as we do not have a expert) represents a hard RL problem and an open research question. Due to time and computational constraints, we have not had the possibility to investigate this direction further. In the conclusions (Chapter 7), we illustrate a few interesting research problems motivated by our findings.

In the remaining of this section, we evaluate the performance of the GA algorithms. In Figure 6.6, we can see an example of what a single GA run looks like on the Sioux Falls network (the most used in traffic research). The algorithms are run for 100 iterations. As we can see, no improvement is obtained by discrete edge removals. This means that the travel time cannot be improved by removing entire edges or combination of edges. The Braess' paradox does not occur in his classical formulation. However, great improvements can be obtained by decreasing capacities. Continuous GA obtains the best performance, improving by 3% the original total travel time and getting very close to the SO on the original network.

Having seen how a single run looks like, we then run each of our three GA formulations for 20 times on each of the six real-world transport networks. Each run lasts for 100 iterations. As the GA is based on probability, running it 20 times is fundamental to get a distribution over the obtained improvements. The results are shown in Figure 6.7. Here, the discrete removal GA formulation is not plotted as it always achieves 0% improvement, this is because, also in these networks, the classical Braess' paradox does not occur

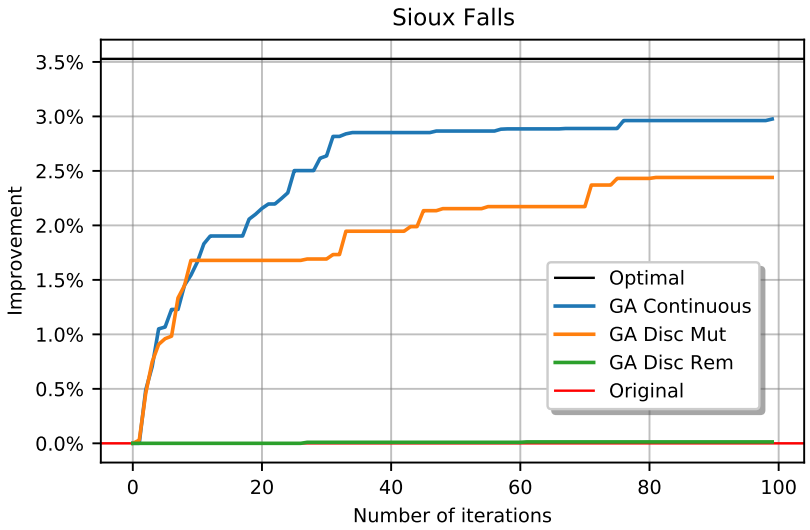


Figure 6.6: Total travel time improvement for the three genetic algorithm formulations in a single run on the Sioux Falls transport network.

and travel time cannot be improved by simple edge removals. However, we see that the other GA formulations are able to achieve great improvements. This suggests that a "continuous Braess' paradox" indeed occurs and is addressable through our capacity reductions.

The results show significant improvements in all networks, with almost optimal results for three out of the six. We see that there is not an overall best solution among the two plotted GA approaches, although the continuous formulation achieves overall better results. The results are also summarised in Table 6.6, where additional information on the running time is available. We also report the improvement of the best run among the two algorithms as "GA Best" and the corresponding total travel time saved in hours (first row) for each traffic hour (the flow time unit). As we can see, the results are really promising, with up to 487 hours of total travel time saved in Chicago (more than 20 days).

Table 6.6: Genetic algorithm total travel time improvement results on six real-world transport networks. For the distributions, we report mean ± standard deviation over 20 runs. The highest mean improvement is shown in bold. The best improvement achieved for each network is reported as GA Best and the corresponding total travel time hours saved per traffic hour are reported as Most time saved.

	Anaheim	Barcelona	Chicago	Massachusetts	Sioux Falls	Winnipeg
Most time saved (h)	4.8	3.7	487.8	0.2	67.1	5.9
Optimal (%)	1.35	2.80	27.66	3.17	3.61	6.45
GA Best (%)	1.32	1.03	7.07	3.11	3.23	2.25
GA Cont (%)	**1.30**±0.01	**0.83**±0.11	3.13±0.48	**3.06**±0.05	**3.09**±0.12	**1.96**±0.15
GA Disc Mut (%)	1.25±0.02	0.77±0.11	**4.33**±1.46	3.04±0.01	2.49±0.15	1.73±0.11
Running time (min)	41.8±2.1	55.1±0.9	83.4±2.1	50.4±0.1	32.8±8.3	56.6±0.9

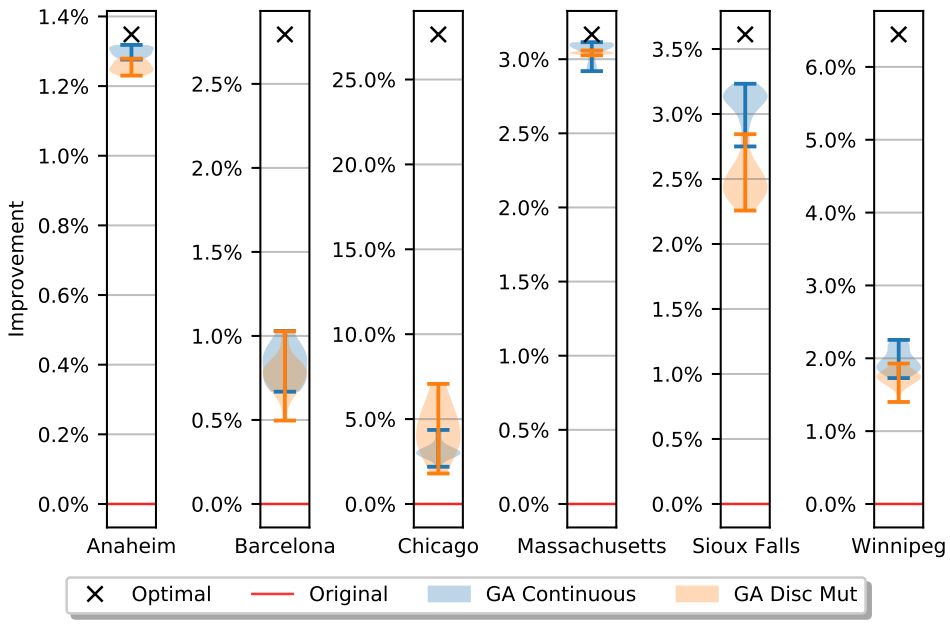


Figure 6.7: Genetic algorithm total travel time improvement results on six real-world transport networks. The violin plots show the distributions over 20 runs.

6.3 Summary

In this chapter, we have presented a series of evaluation experiments. Firstly, we have tested and benchmarked our implementation of self-aware routing, showing that it outperforms state-of-the-art solutions in a range of congestion scenarios. Secondly, we have provided experimental evidence for some of the theoretical results of Chapter 4. Thirdly, we have tested the two network design problems formulated in Chapter 5, showing the great benefit achieved through our solutions. While the RL approach showed that extreme care and research still has to be put into autonomous learning for solving complex optimisation tasks, our genetic algorithm implementations achieve astonishing results on six real-world large-scale transport networks. The relevance of these results is highlighted by the fact that our approach only uses virtual infrastructure reductions, without adding infrastructure or requiring physical modifications to the networks. This makes it ready for immediate deployment on any transport network in the world.

Chapter 7

Conclusions and future work

7.1 Contributions

In this dissertation we have highlighted the importance of transport networks in AVs' routing and showed how we can effectively modify them to increase the global routing performance while remaining fair to the single users. Our main contributions are:

- We proved via counterexample the non-supermodularity of total travel time when paths are added to a transport network with users routed according to SO or UE. Our results are confirmed experimentally through vehicle-level simulations. We also provided and proved particular scenarios in which the property of supermodularity holds.
- We formulated two innovative transport network design problems and framed them in a bilevel optimisation framework. For the first one, we start from a trip spanning tree and seek to optimally add paths to it in order to decrease the total travel time while avoiding Braess' paradox. For the second one, we start from a full transport network and seek to optimally remove infrastructure through virtual road capacity reductions in order to push self-interested vehicles to behave optimally.
- In order to solve the path addition problem, we implemented a greedy algorithm and evaluated it on custom grid networks in the SUMO simulator. Our approach is Braess' paradox-aware and, thus, able to save significant total travel time.
- In order to solve the capacity reduction problem, we implemented a Genetic Algorithm (GA) as well as a Reinforcement Learning (RL) environment and agent and tested them on six real-world transport networks. Near optimal results are achieved in three networks, with up to 487 hours of total travel time saved for each traffic hour in Chicago.
- We built a macroscopic traffic simulator and implemented the Frank-Wolfe traffic assignment solver to compute SO and UE. They are already being used by other researchers as standalone products. We also implemented self-aware routing from the literature and benchmarked it on the microscopic traffic simulator SUMO, showing better than state-of-the-art performance.

7.2 Key insights

- Our non-supermodularity results constitute a valuable insight for transport network designers, as they show that an intuitive property of path additions to transport networks does not hold.
- While the RL task suggests that additional investigation is required as it fails to optimise large networks, the GA approach shows outstanding results, providing empirical support for the non-intuitive concept that significant travel time can be saved just by removing infrastructure. This is not obtained by removing entire roads, as imagined by Braess and investigated in current research, but through virtual continuous capacity reductions.

The promising results of the GA and the limitations shown by the data-driven approach motivate many new interesting research directions, which we will discuss in the next section.

7.3 Limitations and future work

An important aspect that is overlooked in this work is the dynamic and time-dependent nature of trip demands. An extension to our research should address dynamic OD matrices and thus provide time-varying network designs and capacity reductions. This could help address the high traffic variability typical of real-world transport networks.

Data-driven solutions for optimisation problems constitute another promising research direction. Recent extraordinary advancements in RL suggest that this paradigm could have the potential to outperform traditional heuristic solutions. Showing this, was one of the objectives of our project. However, our experiments show that it is very challenging for a single agent to explore a large search space without any expert guidance. This has led us to think about two future directions for this learning task:

- Given the encouraging results for one agent on small networks, we could tackle the problem using multiple agents. A network district is assigned to each agent with the task to optimise that fraction of the network. Agents could then act in a decentralised manner by sharing local information with each other. Communication and data aggregation could be enabled by organising the agents in a hierarchical GNN.
- We believe that the main issue with RL in this task is the absence of expert guidance. To overcome this, GA and RL could be fused to form an hybrid solution. An example, could be using GA results as a starting point for the learning process, leaving to the agent only the last near-optimal part of the search space to explore.

Drawing from the bio-inspiration of evolutionary computation, this last direction almost assumes a philosophical interpretation. Our solutions leverage the ideas of evolution (GA) and learning (RL). One could argue that these two components alone are what characterises human intelligence. The idea of fusing them together constitutes a very exciting and promising direction and could potentially be part of the next big step for artificial intelligence.

Appendix A

Proofs

A.1 Proof of Theorem 4.1

Proof. We can rewrite $\Lambda(\mathcal{B}) \leqslant \Lambda(\mathcal{A})$ as

$$\min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{\mathcal{B} \oplus \mathcal{I}}} c_p x_p \right\} \leqslant \min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{\mathcal{A} \oplus \mathcal{I}}} c_p x_p \right\} \tag{A.1}$$

If $\mathcal{A} \subseteq \mathcal{B}$ there exists $\mathcal{Y}$ s.t. $\mathcal{B} = \mathcal{A} \cup \mathcal{Y}$. Therefore we can rewrite the above equation as

$$\min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{\mathcal{Y} \oplus \mathcal{A} \oplus \mathcal{I}}} c_p x_p \right\} \leqslant \min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{\mathcal{A} \oplus \mathcal{I}}} c_p x_p \right\} \tag{A.2}$$

We know that $P_{\mathcal{Y} \oplus \mathcal{A} \oplus \mathcal{I}}$ can be decomposed into its two components $P_{\mathcal{A} \oplus \mathcal{I}}$ and $P_{\mathcal{Y} | \mathcal{A} \oplus \mathcal{I}}$. Thus we can rewrite

$$\begin{aligned} \min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{\mathcal{A} \oplus \mathcal{I}}} c_p x_p + \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{\mathcal{Y} | \mathcal{A} \oplus \mathcal{I}}} c_p x_p \right\} \leqslant \\ \min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{\mathcal{A} \oplus \mathcal{I}}} c_p x_p \right\} \end{aligned} \tag{A.3}$$

If we turn our attention to the left side of the inequality and particularly to the second term of the sum, we can set $x_p = 0,\ \forall p \in P^m_{\mathcal{Y} | \mathcal{A} \oplus \mathcal{I}},\ \forall m \in \mathcal{M}$; thus obtaining, in this specific class of points

$$\min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{\mathcal{A} \oplus \mathcal{I}}} c_p x_p \right\} = \min_{\mathbf{x}} \left\{ \sum_{m \in \mathcal{M}} \sum_{p \in P^m_{\mathcal{A} \oplus \mathcal{I}}} c_p x_p \right\} \tag{A.4}$$

This confirms our theorem as the minimum of a function in a specific class of points is greater or equal to the minimum of the function in a general point. ■

A.2 Proof of Theorem 4.5

Proof. We begin by recalling the definition of supermodularity. Given two subsets $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{G}''$ and $x \in \mathcal{G}'' \setminus \mathcal{B}$, it holds that

$$\Lambda(\mathcal{A}) - \Lambda(\mathcal{A} \cup \{x\}) \geqslant \Lambda(\mathcal{B}) - \Lambda(\mathcal{B} \cup \{x\}) \tag{A.5}$$

Knowing that $\mathcal{A} \subseteq \mathcal{B}$ we can define $\mathcal{B} = \mathcal{Y} \cup \mathcal{A}$ with $\mathcal{A} \cap \mathcal{Y} = \varnothing$. This yields

$$\Lambda(\mathcal{A}) - \Lambda(\mathcal{A} \cup \{x\}) \geqslant \Lambda(\mathcal{Y} \cup \mathcal{A}) - \Lambda(\mathcal{Y} \cup \mathcal{A} \cup \{x\}) \tag{A.6}$$

Which, according to the definition of Λ and to Equation 4.12, we can rewrite as:

$$\begin{aligned} d\min\{c_p | p \in P_{\mathcal{A}\oplus\mathcal{I}}\} - d\min\{c_p | p \in P_{x\oplus\mathcal{A}\oplus\mathcal{I}}\} \geqslant \\ d\min\{c_p | p \in P_{\mathcal{Y}\oplus\mathcal{A}\oplus\mathcal{I}}\} - d\min\{c_p | p \in P_{x\oplus\mathcal{Y}\oplus\mathcal{A}\oplus\mathcal{I}}\} \end{aligned} \tag{A.7}$$

The term d can be simplified knowing that $d \geqslant 0$. We proceed to consider the following two cases:

Case 1: $\min\{c_p | p \in P_{x\oplus\mathcal{A}\oplus\mathcal{I}}\} \geqslant \min\{c_p | p \in P_{\mathcal{Y}\oplus\mathcal{A}\oplus\mathcal{I}}\}$. This implies that

$$\min\{c_p | p \in P_{\mathcal{Y}\oplus\mathcal{A}\oplus\mathcal{I}}\} = \min\{c_p | p \in P_{x\oplus\mathcal{Y}\oplus\mathcal{A}\oplus\mathcal{I}}\}$$

Equation A.7 becomes

$$\min\{c_p | p \in P_{\mathcal{A}\oplus\mathcal{I}}\} - \min\{c_p | p \in P_{x\oplus\mathcal{A}\oplus\mathcal{I}}\} \geqslant 0 \tag{A.8}$$

By Lemma 4.4 we know that $P_{\mathcal{A}\oplus\mathcal{I}} \subseteq P_{x\oplus\mathcal{A}\oplus\mathcal{I}}$. Recalling that the minimum of a set is always greater or equal than the minimum of one of its supersets, A.8 holds.

Case 2: $\min\{c_p | p \in P_{x\oplus\mathcal{A}\oplus\mathcal{I}}\} < \min\{c_p | p \in P_{\mathcal{Y}\oplus\mathcal{A}\oplus\mathcal{I}}\}$. This implies that

$$\min\{c_p | p \in P_{x\oplus\mathcal{A}\oplus\mathcal{I}}\} = \min\{c_p | p \in P_{x\oplus\mathcal{Y}\oplus\mathcal{A}\oplus\mathcal{I}}\}$$

Equation A.7 becomes

$$\min\{c_p | p \in P_{\mathcal{A}\oplus\mathcal{I}}\} \geqslant \min\{c_p | p \in P_{\mathcal{Y}\oplus\mathcal{A}\oplus\mathcal{I}}\} \tag{A.9}$$

By Lemma 4.4 we know that $P_{\mathcal{A}\oplus\mathcal{I}} \subseteq P_{\mathcal{Y}\oplus\mathcal{A}\oplus\mathcal{I}}$. Recalling that the minimum of a set is always greater or equal than the minimum of one of its supersets, A.9 holds. ■

A.3 Proof of Lemma 4.6

Proof. From the definition of convexity we know that c_e is convex if and only if for all $0 \leqslant t \leqslant 1$ and all $x_1, x_2 \in [0, u_e]$,

$$c_e(tx_1 + (1-t)x_2) \leqslant tc_e(x_1) + (1-t)c_e(x_2) \tag{A.10}$$

By the definition of c_e, we can rewrite

$$\frac{l_e}{v_e^{max}\left(1-\frac{tx_1+(1-t)x_2}{u_e}\right)} \leqslant \frac{tl_e}{v_e^{max}\left(1-\frac{x_1}{u_e}\right)} + \frac{(1-t)l_e}{v_e^{max}\left(1-\frac{x_2}{u_e}\right)} \tag{A.11}$$

Knowing that $l_e \geqslant 0$, $v_e^{max} \geqslant 0$, and $u_e \geqslant 0$, we can apply a series of simplifications and rewrites

$$\frac{u_e}{u_e - tx_1 - (1-t)x_2} \leqslant \frac{tu_e}{u_e - x_1} + \frac{(1-t)u_e}{u_e - x_2} \tag{A.12}$$

$$\frac{1}{u_e - tx_1 - (1-t)x_2} \leqslant \frac{t(u_e - x_2) + (1-t)(u_e - x_1)}{(u_e - x_1)(u_e - x_2)} \tag{A.13}$$

$$\frac{1}{u_e + t(x_2 - x_1) - x_2} \leqslant \frac{u_e + t(x_1 - x_2) - x_1}{(u_e - x_1)(u_e - x_2)} \tag{A.14}$$

Both denominators are greater or equal to zero. We can thus write

$$(u_e - x_1)(u_e - x_2) \leqslant (u_e + t(x_2 - x_1) - x_2)(u_e + t(x_1 - x_2) - x_1) \tag{A.15}$$

$$0 \leqslant t^2(2x_1x_2 - x_1^2 - x_2^2) + t(x_1^2 + x_2^2) \tag{A.16}$$

$$0 \leqslant x_1^2(1-t) + x_2^2(1-t) + 2x_1x_2 \tag{A.17}$$

All the three components on the right hand side are greater or equal to zero and thus the equation holds. ■

A.4 Proof of Theorem 4.16

Proof. As we have shown with Lemma 4.14 and Lemma 4.15 that Λ_U and Λ_O are equivalent on $\mathcal{H}''$, we proceed to prove the theorem for Λ_O, the result will be also valid for Λ_U. We begin by recalling the definition of supermodularity. Given two subsets $\mathcal{A}$ and $\mathcal{B}$ such that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{H}''$ and $x \in \mathcal{H}'' \setminus \mathcal{B}$, it holds that

$$\Lambda_O(\mathcal{A}) - \Lambda_O(\mathcal{A} \cup \{x\}) \geqslant \Lambda_O(\mathcal{B}) - \Lambda_O(\mathcal{B} \cup \{x\}) \tag{A.18}$$

Knowing that $\mathcal{A} \subseteq \mathcal{B}$ we can define $\mathcal{B} = \mathcal{Y} \cup \mathcal{A}$ with $\mathcal{A} \cap \mathcal{Y} = \varnothing$. This yields

$$\Lambda_O(\mathcal{A}) - \Lambda_O(\mathcal{A} \cup \{x\}) \geqslant \Lambda_O(\mathcal{Y} \cup \mathcal{A}) - \Lambda_O(\mathcal{Y} \cup \mathcal{A} \cup \{x\}) \tag{A.19}$$

We know that the trip spanning tree $G_{\mathcal{I}}$ contains only one path $|P_{\mathcal{I}}| = 1$. We call the number of paths in $\mathcal{A}$ and $\mathcal{Y}$, $a \geqslant 0$ and $y \geqslant 0$ respectively. We also know that $|P_{\mathcal{A}\oplus\mathcal{I}}| = a + 1$. We fix $n = a + 1$.

By Lemma 4.15 we know that Equation 4.22 holds, thus, using the definition of Λ_O, we can rewrite

$$dc_p\left(\frac{d}{n}\right) - dc_p\left(\frac{d}{n+1}\right) \geqslant dc_p\left(\frac{d}{n+y}\right) - dc_p\left(\frac{d}{n+y+1}\right) \tag{A.20}$$

By expanding, we have

$$\begin{gathered} d\frac{l}{v^{max}\left(1-\frac{d}{nu}\right)} - d\frac{l}{v^{max}\left(1-\frac{d}{(n+1)u}\right)} \geqslant \\ d\frac{l}{v^{max}\left(1-\frac{d}{(n+m)u}\right)} - d\frac{l}{v^{max}\left(1-\frac{d}{(n+m+1)u}\right)} \end{gathered} \tag{A.21}$$

We know that $d, l, v^{max}, u \geqslant 0$. Thus, we can apply a series of rewrites

$$\frac{n}{nu-d} - \frac{n+1}{(n+1)u-d} \geqslant \frac{n+m}{(n+m)u-d} - \frac{n+m+1}{(n+m+1)u-d} \tag{A.22}$$

$$\frac{1}{(nu-d)((n+1)u-d)} \geqslant \frac{1}{((n+m)u-d)((n+m+1)u-d)} \tag{A.23}$$

Thanks to Property (4) of a *trip spanning tree*, we know that $u \geqslant d$, thus both denominators are non-negative and we have

$$((n+m)u-d)((n+m+1)u-d) \geqslant (nu-d)((n+1)u-d) \tag{A.24}$$

$$mu(mu+u+2nu-2d) \geqslant 0 \tag{A.25}$$

Which, by the fact that $n \geqslant 1$ and $u \geqslant d$, is always true. ■

Bibliography

[1] Dietrich Braess, Anna Nagurney, and Tina Wakolbinger. On a paradox of traffic planning. *Transportation science*, 39(4):446–450, 2005.

[2] Daniel Krajzewicz, Georg Hertkorn, Christian Rössel, and Peter Wagner. Sumo (simulation of urban mobility)-an open-source traffic simulation. In *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*, pages 183–187, 2002.

[3] David James Wilkie, Jur Van den Berg, Ming Lin, and Dinesh Manocha. Self-aware traffic route planning. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[4] Larry J LeBlanc, Edward K Morlok, and William P Pierskalla. An efficient approach to solving the road network equilibrium traffic assignment problem. *Transportation research*, 9(5):309–318, 1975.

[5] Nicola Bellomo and Christian Dogbe. On the modelling crowd dynamics from scaling to hyperbolic macroscopic models. *Mathematical Models and Methods in Applied Sciences*, 18(supp01):1317–1345, 2008.

[6] John Glen Wardrop. Road paper. some theoretical aspects of road traffic research. *Proceedings of the institution of civil engineers*, 1(3):325–362, 1952.

[7] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows. 1988.

[8] Y Sheffi. Equilibrium analysis with mathematical programming methods. *Massachusetts: Institute of Technology*, 1985.

[9] BD Greenshields, JR Bibbins, WS Channing, and HH Miller. A study of traffic capacity. In *Highway research board proceedings*, volume 1935. National Research Council (USA), Highway Research Board, 1935.

[10] Florian Siebel and Wolfram Mauser. On the fundamental diagram of traffic flow. *SIAM Journal on Applied Mathematics*, 66(4):1150–1162, 2006.

[11] Tim Roughgarden and Éva Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49(2):236–259, 2002.

[12] Tim Roughgarden. *Selfish routing and the price of anarchy*. MIT press, 2005.

[13] Tim Roughgarden. The price of anarchy is independent of the network topology. *Journal of Computer and System Sciences*, 67(2):341–364, 2003.

[14] Marguerite Frank, Philip Wolfe, et al. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[15] Christian Gawron. An iterative algorithm to determine the dynamic user equilibrium in a traffic simulation model. *International Journal of Modern Physics C*, 9(03):393–407, 1998.

[16] Richard Steinberg and Willard I Zangwill. The prevalence of braess' paradox. *Transportation Science*, 17(3):301–318, 1983.

[17] Eric I Pas and Shari L Principio. Braess' paradox: Some new insights. *Transportation Research Part B: Methodological*, 31(3):265–276, 1997.

[18] Gregory Valiant and Tim Roughgarden. Braess's paradox in large random graphs. *Random Structures & Algorithms*, 37(4):495–515, 2010.

[19] Jaume Barceló et al. *Fundamentals of traffic simulation*, volume 145. Springer, 2010.

[20] Harold J Payne. Freflo: A macroscopic simulation model of freeway traffic. *Transportation Research Record*, (722), 1979.

[21] Jaume Barceló, Esteve Codina, Jordi Casas, Jaume L Ferrer, and David García. Microscopic traffic simulation: A tool for the design, analysis and evaluation of intelligent transport systems. *Journal of intelligent and robotic systems*, 41(2-3):173–203, 2005.

[22] Wilco Burghout, Haris N Koutsopoulos, and Ingmar Andreasson. A discrete-event mesoscopic traffic simulation model for hybrid traffic simulation. In *2006 IEEE Intelligent Transportation Systems Conference*, pages 1102–1107. IEEE, 2006.

[23] Kay W Axhausen, Andreas Horni, and Kai Nagel. *The multi-agent transport simulation MATSim*. Ubiquity Press, 2016.

[24] Jaime Barceló and Jordi Casas. Dynamic network simulation with aimsun. In *Simulation approaches in transportation analysis*, pages 57–98. Springer, 2005.

[25] G Kotusevski and KA Hawick. A review of traffic simulation software. 2009.

[26] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

[27] Donald M Topkis. *Supermodularity and complementarity*. Princeton university press, 2011.

[28] David E Goldberg. Genetic algorithms in search. *Optimization, and MachineLearning*, 1989.

[29] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. Handbook of evolutionary computation. *Release*, 97(1):B1, 1997.

[30] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[33] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[34] Rick Zhang, Kevin Spieser, Emilio Frazzoli, and Marco Pavone. Models, algorithms, and evaluation for autonomous mobility-on-demand systems. In *2015 American Control Conference (ACC)*, pages 2573–2587. IEEE, 2015.

[35] Federico Rossi, Rick Zhang, Yousef Hindy, and Marco Pavone. Routing autonomous vehicles in congested transportation networks: Structural properties and coordination algorithms. *Autonomous Robots*, 42(7):1427–1442, 2018.

[36] Federico Rossi. *On the interaction between Autonomous Mobility-on-Demand systems and the built environment: Models and large scale coordination algorithms*. PhD thesis, 2018.

[37] Gioele Zardini, Nicolas Lanzetti, Mauro Salazar, Andrea Censi, Emilio Frazzoli, and Marco Pavone. On the co-design of av-enabled mobility systems. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8. IEEE, 2020.

[38] Amanda Prorok. Robust assignment using redundant robots on transport networks with uncertain travel time. *IEEE Transactions on Automation Science and Engineering*, 17(4):2025–2037, 2020.

[39] Tim Roughgarden. Stackelberg scheduling strategies. *SIAM journal on computing*, 33(2):332–350, 2004.

[40] Tim Roughgarden. On the severity of braess's paradox: Designing networks for selfish users is hard. *Journal of Computer and System Sciences*, 72(5):922–953, 2006.

[41] Y Askoura, Jean-Patrick Lebacque, and H Haj-Salem. Optimal sub-networks in traffic assignment problem and the braess paradox. *Computers & Industrial Engineering*, 61(2):382–390, 2011.

[42] Jie Ma, Dawei Li, Lin Cheng, Xiaoming Lou, Chao Sun, and Wenyun Tang. Link restriction: Methods of testing and avoiding braess paradox in networks considering traffic demands. *Journal of Transportation Engineering, Part A: Systems*, 144(2):04017076, 2018.

[43] Saeed Asadi Bagloee, Avishai Ceder, Madjid Tavana, and Claire Bozic. A heuristic methodology to tackle the braess paradox detecting problem tailored for real road networks. *Transportmetrica A: Transport Science*, 10(5):437–456, 2014.

[44] Ron Cocchi, Scott Shenker, Deborah Estrin, and Lixia Zhang. Pricing in computer networks: Motivation, formulation, and example. *IEEE/ACM Transactions on networking*, 1(6):614–627, 1993.

[45] Richard Cole, Yevgeniy Dodis, and Tim Roughgarden. How much can taxes help selfish routing? *Journal of Computer and System Sciences*, 72(3):444–467, 2006.

[46] Marwaan Simaan and Jose B Cruz. On the stackelberg strategy in nonzero-sum games. *Journal of Optimization Theory and Applications*, 11(5):533–555, 1973.

[47] Reza Zanjirani Farahani, Elnaz Miandoabchi, Wai Yuen Szeto, and Hannaneh Rashidi. A review of urban transportation network design problems. *European Journal of Operational Research*, 229(2):281–302, 2013.

[48] Larry J Leblanc. An algorithm for the discrete network design problem. *Transportation Science*, 9(3):183–199, 1975.

[49] Hossain Poorzahedy and Farhad Abulghasemi. Application of ant system to network design problem. *Transportation*, 32(3):251–273, 2005.

[50] Hossain Poorzahedy and Omid M Rouhani. Hybrid meta-heuristic algorithms for solving network design problem. *European Journal of Operational Research*, 182(2):578–596, 2007.

[51] Yafeng Yin. Genetic-algorithms-based approach for bilevel programming models. *Journal of transportation engineering*, 126(2):115–120, 2000.

[52] Mustafa Abdulaal and Larry J LeBlanc. Continuous equilibrium network design models. *Transportation Research Part B: Methodological*, 13(1):19–32, 1979.

[53] Qiang Meng and Hai Yang. Benefit distribution and equity in road network design. *Transportation Research Part B: Methodological*, 36(1):19–35, 2002.

[54] Tianze Xu, Heng Wei, and Guanghua Hu. Study on continuous network design problem using simulated annealing and genetic algorithm. *Expert Systems with Applications*, 36(2):1322–1328, 2009.

[55] Tom V Mathew and Sushant Sharma. Capacity expansion problem for large urban transportation networks. *Journal of Transportation Engineering*, 135(7):406–415, 2009.

[56] Lester Randolph Ford Jr and Delbert Ray Fulkerson. *Flows in networks*, volume 54. Princeton university press, 2015.

[57] Anthony L Peressini, Francis E Sullivan, and Jerry J Uhl Jr. *The mathematics of nonlinear programming*. Springer-Verlag, 1988.

[58] Martin Beckmann, Charles B McGuire, and Christopher B Winsten. Studies in the economics of transportation. Technical report, 1956.

[59] Stella C Dafermos and Frederick T Sparrow. The traffic assignment problem for a general network. *Journal of Research of the National Bureau of Standards B*, 73(2):91–118, 1969.

[60] Hai Yang. Heuristic algorithms for the bilevel origin-destination matrix estimation problem. *Transportation Research Part B: Methodological*, 29(4):231–242, 1995.

[61] Hayssam Sbayti, Chung-Cheng Lu, and Hani S Mahmassani. Efficient implementation of method of successive averages in simulation-based dynamic traffic assignment models for large-scale network applications. *Transportation Research Record*, 2029(1):22–30, 2007.

[62] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR, 2018.

[63] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[64] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[65] Jin Y Yen. Finding the k shortest loopless paths in a network. *management Science*, 17(11):712–716, 1971.