Signalized Flows

Optimizing Traffic Signals and Guideposts and Related Network Flow Problems

Von der Fakultät für Mathematik, Naturwissenschaften und Informatik
der Brandenburgischen Technischen Universität Cottbus

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
(Dr. rer. nat.)

genehmigte Dissertation

von

Diplom-Mathematiker
Martin Strehler
geboren am 24.06.1982 in Karl-Marx-Stadt

Gutachter: Prof. Dr. rer. nat. Ekkehard Köhler

Gutachter: Prof. Dr. rer. nat. Thomas Erlebach

Gutachter: Prof. Dr. rer. nat. Kai Nagel

Tag der mündlichen Prüfung: 2. Februar 2012

Abstract

Guideposts and traffic signals are important devices for controlling inner-city traffic and their optimized operation is essential for efficient traffic flow without congestion. In this thesis, we develop a mathematical model for guideposts and traffic signals in the context of network flow theory. Guideposts lead to *confluent flows* where each node in the network may have at most one outgoing flow-carrying arc. The complexity of finding maximum confluent flows is studied and several polynomial time algorithms for special graph classes are developed. For traffic signal optimization, a *cyclically time-expanded model* is suggested which provides the possibility of the simultaneous optimization of offsets and traffic assignment. Thus, the influence of offsets on travel times can be accounted directly. The potential of the presented approach is demonstrated by simulation of real-world instances.

Zusammenfassung

Vorwegweiser und Lichtsignalanlagen sind wichtige Elemente zur Steuerung innerstädtischen Verkehrs und ihre optimale Nutzung ist von entscheidender Bedeutung für einen staufreien Verkehrsfluss. In dieser Arbeit werden Vorwegweiser und Lichtsignalanlagen mittels der Netzwerkflusstheorie mathematisch modelliert. Vorwegweiser führen dabei zu *konfluenten Flüssen*, bei denen Fluss einen Knoten des Netzwerks nur gebündelt auf einer einzigen Kante verlassen darf. Diese konfluenten Flüsse werden hinsichtlich ihrer Komplexität untersucht und es werden Polynomialzeitalgorithmen für das Finden maximaler Flüsse auf ausgewählten Graphenklassen vorgestellt. Für die Versatzzeitoptimierung von Lichtsignalanlagen wird ein *zyklisch zeitexpandiertes Modell* entwickelt, das die gleichzeitige Optimierung der Verkehrsumlegung ermöglicht. So kann der Einfluss geänderter Versatzzeiten auf die Fahrzeiten direkt berücksichtigt werden. Die Leistungsfähigkeit dieses Ansatzes wird mit Hilfe von Simulationen realistischer Szenarien nachgewiesen.

Acknowledgments

So many people were involved in the development of this thesis. I wish to thank everybody who supported me.

First of all, I thank my supervisor Ekkehard Köhler. I am grateful for his support and encouragement.

Next, I wish to thank Thomas Erlebach and Kai Nagel for taking the assessments of this thesis.

I am indebted to my math teacher Stephan Lamm who encouraged my mathematical interests and prepared me well for mathematical competitions. I thank Christoph Helmberg for raising my interest in applied discrete mathematics during my years as undergraduate and graduate student even though I was not faithful to the 'matrix magic' of semidefinite programming.

I would like to thank Christian Liebchen and Gregor Wünsch who were involved in the first ideas of the cyclically time-expanded network for traffic signal coordination. I also thank Klaus Nökel from ptv AG in Karlsruhe. He and ptv AG supported our project, provided the VISSIM tool for simulation, and gave good hints and suggestions to understand the traffic engineer's point of view. Likewise, I thank Kai Nagel and his group from TU Berlin, who equipped us with MATSim and its traffic signal plug-in. Additionally, I thank Klaus Nökel, Gregor Wünsch, Sándor Fekete, Kai Nagel, and Dominik Grether for providing real-world scenario data and stimulating ideas on traffic models involving traffic signals.

Furthermore, I thank my coauthors Daniel Dressler and Ekkehard Köhler. I am especially indebted to Daniel, who supplied me with good ideas and good coffee. Our joint work on confluent flows filled several dozen black boards.

I also want to thank all members of the research group of Ekkehard Köhler. Alexander Reich and Harald Schülzke provided an excellent working atmosphere. I enjoyed our discussions about discrete mathematics, mathematical riddles and everything under the sun. Moreover, I would like to thank Alex for his careful proofreading of the manuscript. I thank our students Erik Panzer and René Schneider who helped me coding thousands of lines. Diana Hübner was my sure help in cutting through the red tape.

Last, but surely not least, I am grateful to my friends (not necessarily disjoint from persons mentioned above) and my family.

Cottbus, November 2011 Martin Strehler

Contents

Introduction

Urban traffic congestion is increasing day by day. Examples can be found around the world. Growing cities and increasing population doubled the traffic volume in the last two decades in Europe and North America and an even higher rise has to be expected for the urban regions in Asia or South America in the next years. In Germany, the population travels about 1000 billion kilometers every year, and 85 percent of this distance is covered by individual motor car traffic [25, 56]. The city of São Paulo, Brazil, is famous for its record-breaking traffic jams. The 20 million inhabitants own about six million vehicles. On an evening in June 2009, the traffic congestion the city reached a new record of 293 kilometers in total [47]. In August 2010, there was a 60-mile, nine-day traffic jam near Beijing, China, that even made headlines in Europe.

Traffic congestion causes delays which add up to huge costs for society and business. The urban mobility report 2009 [136] states a total loss of 4.2 billion hours and 87.2 billion dollars for the 439 urban areas in the United States in only one year. Wasted fuel of 2.8 billion gallons, noise, and pollution accumulate. A huge problem has to be solved.

But what can mathematics do to support the quest for stress-free, environment-friendly, and safe traveling? In this thesis we will have a close look at two familiar systems for traffic control–*guideposts* and *traffic signals*. If you do not like traffic jams you are invited to read on and find out how an optimization of guideposts and traffic signals can be used to direct and improve inner-city traffic flow.

Guideposts. Guideposts have been used for a long time. They provide guidance, especially in unfamiliar regions. With increasing mobility, we cannot imagine traveling without guideposts and we can find them everywhere. Even modern GPS-based satellite navigation systems can be seen as small, virtual guideposts inside our cars. Everything seems clear–just follow the guideposts!

But we missed an important question. Where should we install these guideposts? And what consequences arise from our choice?

Assume we want to find a certain point of interest in an unfamiliar network. Fortunately, this network is equipped with guidepost pointing towards our destination wherever a routing decision has to be made. Further, we may assume that these guideposts point in an unique direction, i.e. they exclude all but one road at each intersection. Nothing would be more confusing than two guidepost naming the same destination but pointing in two different directions. Most likely, other traffic participants with the same destination will follow these guideposts, too. If we meet one of them, she will make the same routing decisions just like us. Thus, we will travel on the same route until we reach our destination. Consequently, a group of road users starting at the same origin will share the same path, even if there would exist several alternatives. The capacity of this path limits the amount of traffic participants that can reach the common destination. With road users starting all over the network, a bad choice of guideposts may lead to congestion, although the traffic flow in the network is far away from the network's capacity at free route choice.

Traffic signals. With increasing car traffic, traffic signals managing the right of way at intersections became more and more important. However, the *red* traffic light seems to be dominant. But sometimes, we arrive at a traffic signal and it switches to *green* just in time, so that we can go on without stopping. And once in a blue moon[1], we get even four or five green lights in a row.

Such a traffic signal coordination is a difficult task. Of course, coordinating one road in one direction is rather easy, but with traffic in the opposite direction or traffic in a whole street network it becomes considerably harder.

Even more, changing the coordination also means changing travel times. After a while road users will learn about the fastest routes in the network and they will switch to these routes. This new distribution of traffic in the network may completely disturb our fine-tuned coordination.

Obviously, guideposts and traffic signals are important tools for controlling traffic and traffic control is the backbone in the management of traffic flows in our cities. The optimal use of these signals is essential when we are going to resolve traffic congestion. However, it is sometimes not even clear what 'optimal' means in this context. Guideposts are often installed with respect to the shortest distance towards the destination. Their influence on congestion is poorly studied. In contrast, traffic signal coordination has been investigated for a long time and many approaches and models have been proposed. But these models also recommend various definitions of optimality. The two most common objectives are *minimizing the delay/waiting time of vehicles facing red lights* and *minimizing the number of stops*. Furthermore, the majority of the approaches reveal some deficits like unrealistic modeling of inner-city traffic flows or no guarantee for an optimal solution.

Contribution

In this thesis, we tackle traffic congestion with the help of network flow theory from two sides. First, we advance guideposts from a theoretical point of view and introduce *confluent flows*. A flow is called confluent if the flow uses at most one outgoing arc at each node. Unlike previous results we consider heterogeneous arc capacities. We will focus on $\mathcal{NP}$-hardness results for maximum confluent flows, an approximation algorithm for graphs with treewidth bounded by a constant k, and polynomial time-algorithms for special graph classes.

Second, we advance traffic signals. Since this discussion is actually a practical one – most results presented in this part are an outcome of the ADVEST project that emerged

[1]A *blue moon* refers to the third full moon in a season with four full moons. A season with four full moons is very rare, this happens only once every 2 or 3 years.

between BTU Cottbus, TU Berlin, TU Braunschweig, and PTV AG[2] – we start our contribution with a new model for the simultaneous optimization of traffic signal coordination and traffic assignment. This combined approach accounts the feedback between red lights and route choices. We answer the time dependance of traffic signals by a cyclically time-expanded network. This time expansion will also allow capturing several other characteristics of inner-city traffic like platoons of cars and exact arrival times of these platoons at the intersections. Still, viewing inner-city traffic as a periodic process limits the time horizon of the expansion and leads to a compact formulation of the problem as a mixed-integer program. Solving the MIP yields a guarantee or at least bounds for the optimal solution. We investigate our approach with the help of real-world data and state-of-the-art simulation tools.

Outline of the thesis

In Chapter 1 we will fix the notation and terminology and present basic definitions. We assume the reader to be familiar with the basic concepts in graph theory, complexity theory as well as linear programming. However, for later reference and as a short refreshing of knowledge we recall some of the most important facts. For additional information we refer to [2, 66, 100, 122, 147].

In Chapter 2 we will derive the concept for flows in networks with guideposts. For that, we will introduce *confluent flows* and conclude some basic properties, e.g. the underlying tree structure. We will study complexity result for both the *transshipment* and the *maximum flow* variant. We also present *arc-confluent flows* and discuss *cuts* for confluent flows.

Afterwards, we present *polynomial time algorithms* for restricted graph classes, e.g. trees, planar graphs with at most k terminals on the boundary, and graphs without $K_{2,3}$ as a minor in Chapter 3. The relation between confluent flows and trees will lead to a pseudo-polynomial time solution for maximum confluent flows on graphs with *treewidth* bounded by a constant k. We use this result to develop a *fully polynomial time approximation scheme* (FPTAS) for confluent flows on this kind of graphs.

Due to the various approaches for traffic signal optimization we start with a short survey on this topic in Chapter 4. We will use this survey to make the reader familiar with concepts in traffic engineering and with terms related to traffic signals. We will also discuss the advantages and disadvantages of the considered approaches to motivate our new model. Herewith, the ground for the next two chapters should be prepared. Additional information can be found, e.g., in [69, 141].

In Chapter 5 we use the concept of dynamic flows and the periodicity of traffic signals to develop a cyclically time-expanded network. The model is completed by modeling intersections, traffic signals and traffic assignment. As a main result of this chapter we show how this model can be used to optimize traffic signal coordination and traffic assignment simultaneously. Aiming for a realistic modeling we also discuss the conse-

[2]PTV AG is a traffic planning company from Karlsruhe, Germany. It is well known for its traffic planning and simulation software VISUM and VISSIM.

quences of our approach to travel times and link performance in detail and derive further properties of the model.

Finally, Chapter 6 is designated for the simulation of inner-city traffic and the practical evaluation of the proposed model. We introduce the reader to two traffic simulation tools, namely VISSIM and MATSim. In detail, we consider the real-world inner-city networks of Cottbus, Braunschweig, Portland, and Denver. In particular, we emphasize the advantages of our simultaneous optimization of signal coordination and traffic assignment by comparing to a decomposed successive version of our approach.

About this thesis

A lot of results in this thesis were obtained during the ADVEST project, granted by the German ministry of education and research (BMBF). This also reflects in the thesis. First, some results were already published, see [52, 53, 96, 97, 98]. Second, the aims of the project lead to different kinds of results. On the one hand confluent flows were studied theoretically and are better understood now. But due to the combinatorial complexity practical applicability is – in the moment – poor. One the other hand, in a more experimental approach, a new model for simultaneous traffic signal coordination and traffic assignment was created, implemented, improved and tested with help of simulation tools. Hereby, a model of high practicability was developed, but it is difficult to prove the impact of the model also mathematically. Hence, the first part of the thesis will perhaps be more interesting for readers who focus on combinatorial optimization. The second part may more appeal to readers who are interested in the modeling of real world problems.

1 Basic Definitions and Notation

In this chapter we introduce and fix the basic notation for this thesis. Many fields of discrete mathematics are touched. First, we introduce the graph notation and we present some classical graph problems that we will refer to later. Due to the wide area of graph theory this description cannot be complete. For an introduction to graph theory we suggest, e.g., [152] or [49]. A good textbook on network flows is, for example, [2]. Good textbooks, covering network flows and other combinatorial optimization strategies, are [36, 100, 138]

Furthermore, we fix the notations for algorithms, complexity and approximation. Again, we can only give a short overview. For additional information, we refer to [66] and [9].

Linear Programming and *Integer Programming* are two basic approaches to solve network flow problems and combinatorial optimization problems. We will introduce both techniques in section 1.4 and suggest [147, 153] for further reading.

Please note that the following chapters also provide their own introductions and terms specific to these chapters are defined there.

1.1 Graphs

In this work we consider finite graphs $G = (V, E)$ where $V = V(G)$ is the *vertex set* and elements $v \in V$ are called *vertices* or *nodes*. $E = E(G)$ is the *edge set* of G. We consider both *undirected* and *directed* graphs (*digraphs*). In the case of undirected, loop free graphs the edge set is a subset of V^2, i.e. $E \subseteq \{\{u, v\} : u, v \in V,\ u \neq v\}$.

To denote directed edges, we also call them *arcs*. E is termed *arc set* A. A consists of ordered pairs of nodes, i.e. $A \subseteq V \times V = \{(u, v) : u, v \in V\}$. Therefore, each arc $a \in A,\ a = (u, v)$ is directed from its $\mathrm{tail}(a) = u$ to its $\mathrm{head}(a) = v$. For $v \in V$, we use $\delta^-(v) = \{a \in A :\ v = \mathrm{head}(a)\}$ for the set of incoming arcs and $\delta^+(v) = \{a \in A :\ v = \mathrm{tail}(a)\}$ for the set of outgoing arcs. A graph is called *bi-directed* if it contains for each arc $a = (u, v)$ also the arc in the opposite direction $a' = (v, u)$. A directed graph can be made undirected by simply deleting the directions of the arcs. To make an (undirected) graph a bi-directed one we add both directions for each edge/arc.

The cardinalities of the node and edge sets are denoted by $n = |V|$ and $m = |E|$. A graph with n vertices that contains all possible edges, is called a *complete graph* and denoted by K_n. Obviously, the complete graph has $m = \binom{n}{2} = \frac{n(n-1)}{2}$ edges. Sometimes, a graph is allowed to contain *multi-edges*, i.e., parallel edges. Hence, E, or A respectively, is defined as a multi-set in this case and G is called *multi-graph.* The *induced subgraph* on a vertex set $V' \subseteq V$ is denoted by $G[V']$. The induced arc set of $G[V']$ is denoted by $A[V']$.

A sequence $W = (a_1, \ldots, a_k)$, $a_i \in A$, of arcs is called a *walk* if it fits head to tail, i.e. $\mathrm{head}(a_i) = \mathrm{tail}(a_{i+1})\ \forall i \in \{1, \ldots, k-1\}$. For short, we will use $\mathrm{tail}(W) := \mathrm{tail}(a_1)$ and $\mathrm{head}(W) := \mathrm{head}(a_k)$. $V[W]$ is used for the set of vertices that occur in the arcs of W. To simplify matters, we use $a \in W$ to denote that the arc a is contained in the sequence of arcs in walk W. A *path* P is a walk which passes through every vertex at most once. A walk/path where the tail of the first arc and the head of the last arc coincide is called *cycle/circuit*. For $u, v \in V$, $\mathcal{P}_{u,v}$ denotes the set of all paths with tail u and head v. The *length* of a path (with respect to *unit edge lengths*) is the number of arcs in its sequence. Two paths P_1 and P_2 are (arc) *disjoint* if $A[V[P_1]] \cap A[V[P_2]] = \emptyset$. They are *node disjoint* if $V[P_1] \cap V[P_2] = \emptyset$. The composition of two paths $P_1 = (a_1, \dots, a_k)$ and $P_2 = (b_1, \dots, b_l)$ is defined as $P_1 \circ P_2 = (a_1, \dots, a_k, b_1, \dots, b_l)$.

Similarly, walks, paths, cycles, and circuits can be defined for undirected graphs.

A graph is *strongly connected* if for each pair of vertices (u, v) there exists a path P from u to v, i.e. $\text{tail}(P) = u$ and $\text{head}(P) = v$. A graph is *weakly connected* if its corresponding bi-directed graph is strongly connected. A node set $U \subseteq V$ is (strongly/weakly) connected if the induced graph $G[U]$ is (strongly/weakly) connected. The inclusion maximal (strongly/weakly) connected subgraphs of G are called *(strongly/weakly) connected components*.

A *cut* in G is an arc set C such that $G \backslash C = (V, A \backslash C)$ has at least one connected component more than G. The *value* of a cut is simply the number of arcs in the cut. An *s-t-cut* is defined as a partition of V into two subsets C_1 and C_2, such that $s \in C_1$ and $t \in C_2$. Let $C = \{a \in A : \text{tail}(a) \in C_1 \wedge \text{head}(a) \in C_2\}$ then there exists no directed path from s to t in $G \backslash C$. The value of the *s-t*-cut is $|C|$, i.e. the number of forward arcs from C_1 to C_2.

1.2 Flows and Networks

Flows have been a major planning tool for many applications ever since Ford and Fulkerson [58] studied them. Before considering flows with additional routing constraints in the following sections, standard flow is introduced here.

1.2.1 Definitions

Although there is consens what a *flow* in a *network* should be, we will use a slightly different and modular approach for defining flows[3].

A *flow function* is a non-negative function on the arc set $x : A \to \mathbb{R}_0^+$. In most practical applications the maximal flow on each arc is limited by a *capacity* bound, i.e., a maximal flow value that cannot be exceeded on this arc. The capacities are given by a function $u : A \to \mathbb{R}_0^+ \cup \{\infty\}$. A flow is *feasible* if $0 \leq x(e) \leq u(e)\ \forall e \in A$ holds. A graph together with capacities is called a *network* $G = (V, A, u)$. For some results in this thesis, we will limit the capacity function to integer values, i.e., $u : A \to \mathbb{N}_0$. This is no restriction for most applications, since rational values can simply be scaled to integer values and irrational numbers cannot exactly be represented in our computers anyway[4].Furthermore, a flow function is *integral* if it has only integral values.

[3]In some of the algorithms especially in Section 3, we will not be able to fulfill all requirements of a flow at once. A modular definition admits step-by-step procedures. For example, we will define and derive *preflows* with a slightly different flow conservation constraint.

[4]Irrational input may yield a unexpected behavior, the algorithm of Ford and Fulkerson is a prime example.

The *inflow* and *outflow* of a flow function x at a given node v are

$$\mathrm{inflow}_x(v) := \sum_{a \in \delta^-(v)} x(a)$$

$$\mathrm{outflow}_x(v) := \sum_{a \in \delta^+(v)} x(a)$$

and the *balance* of a flow function x at a node v is defined as net flow out of this node

$$\mathrm{bal}_x(v) := \mathrm{outflow}_x(v) - \mathrm{inflow}_x(v).$$

We can now distinguish between three kinds of nodes. At some nodes $v \in V$ flow may enter into the network, i.e., the outflow is higher than the inflow. Therefore, these nodes have positive balance and we call them *sources*. The set of all sources is denoted S^+. At other nodes, the balance may be negative and flow may leave the network. These nodes are called *sinks* and the set of all sinks is accordingly denoted S^-. All nodes in S^+ and S^- are also called *terminals* and we demand $S^+ \cap S^- = \emptyset$. For all other nodes v we require that x satisfies the *flow conservation* constraint, i.e., $bal_x(v) = 0$.

Definition 1.1 (Flow). *A* flow *in a network $G = (V, A, u)$ with sources S^+ and sinks S^- is a feasible flow function $x : A \to \mathbb{R}_0^+$ that*

1. *satisfies the flow conservation at all non-terminals,*
2. *has non-negative balance at all sources,*
3. *has non-positive balance at all sinks.*

An *s-t-flow* is a flow with a single source $s \in V$ and a single sink $t \in V$. A *circulation* is a flow where $bal_x(v) = 0$ holds for all $v \in V$ and thus, there are no sources or sinks.

The *value* of a flow x is the amount of flow reaching the sinks, i.e.,

$$val(x) = -\sum_{v \in S^-} bal_x(v).$$

This is equal to $val(x) = \sum_{v \in S^+} bal_x(v)$ due to flow conservation at the non-terminals. For most applications we need to restrict the balances of the terminals. We use a *supply/demand function* $d : V \to \mathbb{R}$. All sources have positive values $d(v) > 0$, while all sinks have negative values $d(v) < 0$. For non-terminals we require $d(v) = 0$. When we talk about the *supply* of a source or the *demand* of a sink this refers to the absolute values of the supply/demand function.

We can now study two variants of flow problems. In the case of a *transshipment* the supplies and demands should exactly be *satisfied.* This means $bal_x(v) = d(v)\ \forall v \in V$. Of course, this can only be achieved if $\sum_{v \in V} d(v) = 0$. In a weaker variant a source s may send up to $d(s)$ and a sink t can accept up to $d(t)$ units of flow. A flow *obeys* the supply/demand function if $0 \le bal_x(v)d(v) \le d(v)^2\ \forall v \in V$.

These two variants lead to two problems.

Problem 1.2. *Consider a network $G = (V, A, u)$ with sources S^+, sinks S^-, and a matching supply/demand function d.*

1. *Is there a flow x that satisfies d? (*Transshipment Problem*)*
2. *What is the maximum value of a flow x obeying d? (*Max Flow*)*

Note that both problems can also be formulated as a circulation problem. In many problems one can typically add a *supersource* s^* and a *supersink* t^* without changing the problem. More precisely, one connects the supersource with all sources, i.e., one adds the arcs $(s^*, s)\ \forall s \in S^+$ with capacities $u((s^*, s)) = d(s)$. Respectively, the same is done for the supersink with arcs $(s, t^*)\ \forall s \in S^-$ with capacities $u((s, t^*) = -d(s)$. Connecting supersink and supersource as well, we require $x((t^*, s^*)) = \sum_{v \in S^+} d(v)$ for the transshipment or we try to maximize $x((t^*, s^*))$ for a maximum circulation. Furthermore, flow conservation applies in every node.

This general definition of flows can be extended by adding *costs* to each arc, i.e., there is another function $c : A \to \mathbb{R}_0^+$. These *arc costs* can be interpreted as the length of an arc or a toll that has to be paid for using this arc. The total cost of flow on an arc e is $f(e)c(e)$ and the total cost of a flow in a network is given by $\sum_{e \in A} c(e)f(e)$. We can now vary the above flow problems by adding an additional constraint, which limits the overall costs of a flow. Or we can look for the cheapest flow among all flows with maximum flow value.

Problem 1.3. *Consider a network $G = (V, A, u)$ with sources S^+, sinks S^-, a matching supply/demand function d, and a cost function c.*

1. *What is the minimum cost of a transshipment? (*Min Cost Transshipment*)*
2. *What is the minimum cost of a maximum flow? (*Min Cost Flow*)*

One can also think of negative costs, i.e., getting money for using an arc. But this complicates the computation of shortest paths and Min Cost Flow. Shortest paths can have negative infinite length if the graph contains a cycle with negative cost sum. On the other hand a minimum circulation can be used to calculate a Max Flow. Just add supersource and supersink as above with $c((t^*, s^*)) = -1$, and all other costs $c(e) = 0,\ e \neq (t^*, s^*)$.

Up to now, we considered only homogeneous flows. In many applications one has to deal with several goods with different origins and destinations in the same network. This can be modeled by *multicommodity flows*. For each commodity $i, i \in \{1, \ldots, N\}$ we implement a separate flow function $x_i : A \to \mathbb{R}_0^+$. We require flow conservation as above for each flow x_i. The capacity of an arc is shared among the commodities, $\sum_{i=1}^N x_i(e) \leq u(e)\ \forall e \in A$. All problems given above can be formulated in a multicommodity variant [2].

1.2.2 Important results

In 1927, Menger presented his famous result which shows the relationship between disjoint paths and cuts.

Theorem 1.4 (Menger, 1927). *Let $G = (V, A)$ be a graph and $s, t \in V$. The maximum number of edge disjoint paths from s to t is equal to the minimum value of an s-t-cut.*

Introducing arc weights, i.e. arc capacities, this result can be extended to flows. Hereby, the *capacity* of an s-t-cut is $\sum_{e \in C} u_e$ where C is the set of forward arcs from $C_1 \ni s$ to $C_2 \ni t$.

Theorem 1.5 (Maximum-flow Minimum-cut). *The maximum value of an s-t-flow is equal to the minimum capacity of an s-t-cut.*

This theorem was proven by P. Elias, A. Feinstein, and C.E. Shannon in 1956, and independently also by L.R. Ford, Jr. and D.R. Fulkerson in the same year.

Several algorithms for finding a maximum s-t-flow have been developed since the 1950s. To name a few approaches, Ford and Fulkerson constructed an algorithm that is related to their constructive proof of the maximum-flow minimum-cut theorem. This algorithm uses *augmenting* paths and was improved by Dinic and by Edmonds and Karp. Goldberg and Tarjan presented a push-relabel-algorithm, which was improved by Ahuja and Orlin. Note that, despite the bunch of combinatorial algorithms for s-t-flows, no exact combinatorial algorithm is known for multicommodity flows, yet.

The successively increasing flow value in the algorithm of Ford and Fulkerson also leads to an important observation.

Corollary 1.6 (Integrality theorem). *If the capacity function u is integral, there exists an integral maximum flow.*

Note that this result is not true for multicommodity flows.

There is also another important link between flows and paths. Let x be an s-t-flow in a network $G = (V, A, u)$ with $s, t \in V$. Then there exists up to $m = |A|$ s-t-paths or cycles, such that x can be represented as a nonnegative linear combination of these paths or cycles. Moreover, if x is integral, the linear combination can be realized with integral coefficients. This result is also known as *path decomposition*.

1.3 Algorithms and Complexity

In the previous section we just mentioned some algorithms for maximum flows and many more algorithms for restricted flows will follow in this thesis. But when talking about algorithms, we have to discuss *running times* and *complexity*. Again, we can only provide an overview.

An *algorithm* is a finite list of instructions. These instructions perform operations on the given data, but they need not be performed in linear order. Each instruction also determines which instruction is next or may even stop the algorithm. Therefore, the algorithm itself is fixed, but the input data may vary.

Obviously, the running time of an algorithm may depend on the size of the input. We will not discuss memory models or machine models here. For a detailed survey on deterministic Turing machines see [66].

For short, we measure the size of the input data as its lengths in some *binary encoding*. The running time of an algorithm is measured in the number of *elementary arithmetic operations* (addition, subtraction, multiplication, division, comparison), that are performed until the algorithm stops. Even inputs of the same size may lead to different behavior of the algorithm. Considering the worst case, the *time complexity function* for an algorithm is the largest amount of time for each possible input length, that is needed by the algorithm to solve an instance of this size.

1.3.1 Polynomial-time algorithms

If the number of elementary operations is bounded by a polynomial in the input size, the algorithm is called *polynomial-time algorithm* or *efficient algorithm*. We can describe the running time of an algorithm with help of the $\mathcal{O}$-notation. A function $f(n)$ is in $\mathcal{O}(g(n))$ if there exists a constant factor c and $n_0 \in \mathbb{N}$ such that $|f(n)| \leq c|g(n)|\ \forall n \geq n_0$. Therefore, an algorithm is efficient, if its time complexity function is in $\mathcal{O}(p(n))$ for a polynomial p and input length n.

To simplify matters we do not measure the size of a graph or network in some binary encoding. The size of a graph depends on the number n of nodes and the number m of edges/arcs. Therefore, the input size for graph related problems is in most cases $n+m$.

With help of this notation, we can now compare algorithms. For example, the algorithm for Max Flow from Edmonds and Karp with shortest augmenting paths has a time complexity of $\mathcal{O}(nm^2)$, while the algorithm from Goldberg and Tarjan has time complexity $\mathcal{O}(nm\log(n^2/m))$. Remember that $m < n^2$.

Note that huge constant factors are maybe ignored in this run time analysis. Also, the running time may depend on some additional implementation tricks. The popular Dijkstra's algorithm for shortest path calculation (cf. [50]) can simply be implemented with a time complexity of $\mathcal{O}(n^2)$. Using a more sophisticated data structure, namely Fibonacci heaps, the time complexity can be reduced to $\mathcal{O}(n\log n + m)$ [2].

Sometimes, an input value itself appears in the running time of an algorithm instead of its logarithm (its size). If the complexity bound is also a polynomial of this parameter, the algorithm is called *pseudo-polynomial*.

For other problems, it is useful to describe the input with several parameters. This allows a finer analysis of their inherent complexity. Not only the size of the input data influences the running time of the algorithm, but the structure of the data may be important as well. Therefore, for some hard problems one can compute an answer in a time that is polynomial in the size of the input and exponential in a parameter k. If k is small and fixed, then such problems can still be considered manageable. The corresponding algorithms are called *fixed-parameter tractable*.

1.3.2 $\mathcal{P}$, $\mathcal{NP}$, and co-$\mathcal{NP}$

For some problems one has not found polynomial time algorithms yet. This leads to the question whether some problems are harder than other problems. The classes $\mathcal{P}$, $\mathcal{NP}$, and co-$\mathcal{NP}$ are collections of decision problems. A decision problem is a problem that can be answered by Yes or No. More precisely, we consider a finite alphabet Σ and the set Σ^* of all finite words of letters from Σ. A problem Π is an arbitrary subset of Σ^*. For a given $x \in \Sigma^*$ we have to decide whether $x \in \Pi$.

Problems in $\mathcal{P}$ are considered to be 'easy' problems, because they can be solved on a deterministic Turing machine in polynomial time with respect to the length of the input.

A problem is in $\mathcal{NP}$ when it can be answered on a nondeterministic Turing machine in polynomial time. Clearly, $\mathcal{P} \subseteq \mathcal{NP}$. An equivalent definition for $\mathcal{NP}$ requires that a given certificate (Yes-solution) can be verified on a deterministic Turing machine in polynomial time. For example, it is hard to decide whether a graph contains a Hamiltonian circuit, i.e., a cycle that visits all nodes exactly once. But when a cycle is given, it is easy to check, whether it is a Hamiltonian circuit.

A problem is in co-$\mathcal{NP}$ when its complement is in $\mathcal{NP}$. It holds $\mathcal{P} \subseteq \mathcal{NP} \cap \text{co-}\mathcal{NP}$. But it is an open problem whether $\mathcal{P} = \mathcal{NP}$ or $\mathcal{P} \neq \mathcal{NP}$. Assuming the second case holds, $\mathcal{NP}$-hard problems are not solvable in an acceptable time.

1.3.3 $\mathcal{NP}$-complete problems

A problem in $\mathcal{NP}$ is said to be $\mathcal{NP}$-complete if each problem in $\mathcal{NP}$ can be reduced to this problem. Hereby, a reduction is a polynomial time algorithm that transforms a problem into another problem such that both problems have the same answer. Hence, $\mathcal{NP}$-complete problems are also referred as the hardest problems in $\mathcal{NP}$. A problem is $\mathcal{NP}$-hard if and only if there is an $\mathcal{NP}$-complete problem that is reducible in polynomial time to this problem. In other words, a problem is $\mathcal{NP}$-hard when it is at least as hard as the hardest problems in $\mathcal{NP}$. Note that an $\mathcal{NP}$-hard problem needs not to be in $\mathcal{NP}$. It could be even harder, undecidable or it may not even be a decision problem. This notation dates back to Cook, who proved that there exists a formal language with $\mathcal{NP}$-complete problems [35]. Building on that, Karp [85] showed for 21 popular combinatorial problems (e.g. the Travelling Salesman Problem (TSP)) that they are $\mathcal{NP}$-complete.

1.3.4 Complexity of optimization problems

We defined complexity for decision problems, i.e., problems with a Yes or No answer. But in most problems introduced up to now like Max Flow or Min Cost Flow, we are looking for a minimum or maximum of some objective.

Assume, we minimize a function $f(x)$ over $x \in X$. We consider the following decision problem:

Given a value v, is there an $x \in X$ with $f(x) < v$?

We perform a binary search to find the optimal value v. If we have an upper bound on the size of the optimal solution and a polynomial time algorithm for the decision problem, this usually yields a polynomial time algorithm for the optimization problem. On the other hand, if an optimization problem has an $\mathcal{NP}$-complete decision version, then it is $\mathcal{NP}$-hard.

1.3.5 Approximation

Many practical optimization problems are $\mathcal{NP}$-hard problems. Theory states that most real world instances of such problems cannot be solved exactly in an acceptable time, assuming $\mathcal{P} \neq \mathcal{NP}$. But on the other hand, this is not necessary for a lot of applications and an exact solution is of limited use when, e.g., the input data is noisy. In this case a reasonable good solution near to the optimum is sufficient.

A p-approximation algorithm is an algorithm that runs in polynomial time and calculates a solution that is at most a factor p away from the optimum. The factor p is called the *performance ratio* of the approximation. Obviously, to performance ratio for a maximization algorithm is less than 1, while it is greater than 1 for minimization problems.

If we can find polynomial time approximation algorithms with a performance ratio of $p = (1 + \epsilon)$ for minimization problems or $p = (1 - \epsilon)$ for maximization problems $\forall \epsilon \in (0, 1)$, we call the family $\{\mathcal{A}_\epsilon\}_{0<\epsilon<1}$ a *polynomial time approximation scheme* (PTAS). Furthermore, when each algorithm $\mathcal{A}_\epsilon$ in this family has a running time polynomial in the input size and polynomial in $1/\epsilon$ we call the family $\{\mathcal{A}_\epsilon\}_{0<\epsilon<1}$ a *fully polynomial time approximation scheme* (FPTAS).

Approximation algorithms are not only used for hard problems. For example, there exist combinatorial approximation algorithms for MULTICOMMODITY FLOW (see, e.g., [67]).

1.4 Linear Programming and Integer Programming

1.4.1 Linear programs

Linear programming (LP) is a powerful optimization tool, which we will also use in this thesis. Again, we can only sketch the main ideas here. Fortunately, there exist very good text books and we recommend, e.g., [147, 153].

Whenever we can formulate a problem in the form

$$\begin{aligned} \min \quad & c^\mathsf{T} x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{Q}^n \end{aligned}$$

with the objective $c^\mathsf{T} x$ ($c \in \mathbb{Q}^n$), and some constraints $Ax \leq b$ ($A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^m$) linear programming is an alternative option to solve this problem. For example, the MAX FLOW problem can be formulated as a linear program. For simplicity, we assume a single source s and a single sink t with infinite capacities:

$$\begin{aligned}
\max \quad & \sum_{e=(s,v)} x(e) - \sum_{e=(v,s)} x(e) && && (1)\\
\text{s. t.} \quad & u(e) - x(e) \geq 0 && \forall e \in A && (2)\\
& \sum_{e=(u,v)} x(e) - \sum_{e=(v,w)} x(e) = 0 && \forall v \in V \backslash \{s,t\} && (3)\\
& x(e) \geq 0 && && (4)
\end{aligned}$$

Hereby, the objective (1) is the outgoing flow of the source. The constraints (2) ensure the capacity bounds. The flow conservation is formulated in (3), i.e., the balance is equal to zero. Of course, all flow values have to be non-negative (4).

Linear programs can be solved by the simplex algorithm, presented by Dantzig [44] in 1947. Despite the simplex algorithm has no polynomial running time in theory, it works well in practice.

In 1979, Khachiyan proved that linear programs can be solved in polynomial time with the ellipsoid method [87]. Therefore, whenever we can formulate a problem Π as a linear program, and this formulation is polynomial in time and size of the original problem, this proves that Π is in $\mathcal{P}$. For example, no combinatorial algorithm is known for the MULTICOMMODITY FLOW problem. But it is easy to formulate this problem as a linear program similar to the formulation of MAX FLOW above. Hence, MULTICOMMODITY FLOW is in $\mathcal{P}$.

Interestingly, the ellipsoid method is inefficient in practice. Today, most solvers for LP also use interior points methods, introduced by Karmarkar [84] in 1984, which are efficient in theory and practice.

The existence of a dual program is an important property of linear programs. As a consequence of Farkas' lemma, each linear program has a dual linear programming formulation and both problems have the same optimal objective value [147]. For example, the path based MAX FLOW can be formulated as linear program as follows:

$$\begin{aligned}
\max \quad & \sum_{P \in \mathcal{P}_{s,t}} x_P \\
\text{s. t.} \quad & \sum_{P : e \in P} x_P \leq u(e) && \forall e \in A\\
& x_p \geq 0 && \forall P \in \mathcal{P}
\end{aligned}$$

Note that the set $\mathcal{P}_{s,t}$ is very large in general. It is not a good idea to list all paths between two nodes. However, this formulation is very useful when only considering a rather small set of active paths. This can be achieved, e.g., by a column generation approach. The dual formulation is:

$$\begin{aligned} \min \quad & \sum_{e \in A} y_e u(e) & \\ \text{s. t.} \quad & \sum_{e \in P} y_e \geq 1 & \forall P \in \mathcal{P} \\ & y_e \geq 0 & \forall e \in E \end{aligned}$$

It is not obvious that this dual linear program always has an optimal solution with integer values for y_e. One will need a result concerning total unimodular matrices to prove this. So, assume, we have found this integer solution. This optimal solution of the dual determines a minimum s-t-cut in the graph, i.e., the set of arcs in the cut is $\{e : y_e = 1\}$. All paths between s and t are cancelled by deleting this set of arcs. Furthermore, the objective is the weighted sum over this set of arcs.

Solving primal and dual LP simultaneously yields bounds for the optimal solution. Furthermore, if both solutions have equal value, then optimality is proved.

1.4.2 Integer programs

If some of the variables have to be integral, finding an optimal solution is much harder. Simply dropping the integrality constraints and rounding does not need to yield good solutions in general. In fact, a special case, the decision version of an integer program with binary variables, is one of Karp's 21 $\mathcal{NP}$-complete problems.

Integer Programming and Mixed Integer Programming are based on linear programming, but they use various strategies to handle integrality constraints. Typically, one solves the relaxation of the problem, i.e., the integrality constraints are dropped. If the solution is not integral, the problem is modified and solved again. These modifications include adding new constraints or splitting the problem. Two strategies are briefly introduced here. For further reading, we recommend [153].

Cutting plane methods This method was first used by Dantzig et al. [45] to solve an instance of the Travelling Salesman Problem, containing 49 major cities in the United States. Gomory [70] generalized this approach to arbitrary integer programs. A cutting plane is an additional constraint that refines the relaxation of an integer program. If the optimal solution of the relaxation is not integral, then there exists a linear inequality separating the optimum from the integer feasible set. This inequality can be added to the relaxation. Obviously, the current optimum is no longer feasible. This process is repeated until an optimal integer solution is found.

Branch&Bound The branch&bound method was introduced by Land and Doig [104]. This enhanced enumeration method consists of two steps. In the branching step the problem is split into two smaller problems. For example, assume the value of a variable x is not integral in the optimal solution, i.e. $x = r$ and $r \notin \mathbb{Z}$. We may now consider two new problems:

- one with the additional constraint $x \leq \lfloor r \rfloor$
- the other one with the additional constraint $x \geq \lceil r \rceil$

Obviously, the union of these two sub-problems is the original problem. Now, a recursive branching is executed. In each step, a new variable and a new threshold is chosen, depending on the branch in the last step. In the bounding step upper and lower bounds for each sub-problem are calculated. If we consider a minimization problem and a lower bound of some sub-problem X is greater than the upper bound of another sub-problem Y, then X can be safely discarded from the branching (pruning), since Y will always yield better solutions. Again, this is repeated until an optimal integer solution is found.

2 Destination-based Routing and Confluent Flows

2.1 Guideposts, destination-based routing, and confluent flows

2.1.1 Guideposts

Guideposts provide orientation for travelers in unfamiliar regions. The oldest known guideposts which can still be found are probably the milestones of the Roman Empire all over Southern Europe. When visiting Rome and the *forum romanum* one can take a look at the basement of the *Milliarium Aureum* (golden milestone). All milestones in the Roman Empire provided the distance and direction to this central point. Most likely, this is the origin of the saying ‘all roads lead to Rome’. Another kind of guideposts still used today are *fingerposts*. Such fingerposts dating back to the 17th century can still be found in England, but they have surely been used much earlier.

Today, modern guideposts, e.g. the one in Figure 1, can be found nearly everywhere. There are 3.5 million traffic guideposts only in Germany, equating to one guidepost every 160 metres in average. But guideposts are not only used for traffic guidance. Some other applications, e.g. designing emergency exit plans, are discussed in Section 2.1.3.

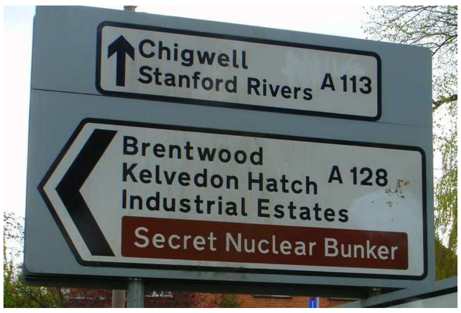

Figure 1: Guidepost near Kelvedon Hatch, North-east of London.

2.1.2 Destination-based Routing

The guidepost in Figure 1 is confusing in a funny way. But there is an even simpler scenario for confusing guideposts: two guideposts at the same intersection, naming the same destination, but pointing in different directions. Thinking of emergency exit routes this could be dangerous as a person fleeing in panic may not act logically.

Of course, we want to avoid confusing guideposts. Therefore, we make the following assumption. Whenever representing a street network or a building as a graph, we allow at most one guidepost for every destination per node. This implies that the chosen path only depends on the destination. This also provides a very easy routing protocol for flow units representing traffic participants. Whenever two flow units with the same destination meet they will stay together for the remaining journey. If the guideposts are already installed, routing itself is very easy. *Placing* the guideposts is the main task.

Before we are delving into theory we will take a closer look at some applications.

2.1.3 Applications

Besides common road traffic, destination-based routing is used in a variety of other applications. In practice, a simple routing protocol is often desired for various reasons. Sometimes, the demand distribution in the network is not known a priori and the routing decision has to be made locally. In other cases, there is not enough time for calculating another routing protocol which is optimal with respect to a more sophisticated objective function.

Evacuation. Most of the time, we do not pay much attention to these green pictograms with the little stick figure fleeing from danger: *emergency exit signs*. Creating evacuation plans means, besides lots of other tasks, choosing the escape routes. And here a crucial point arises. Two persons arriving at the same evacuation sign will follow the same route.

Herd behavior has been observed in many species. When panicked individuals can choose between two equal exits of a room, most of them tend to stay together. The majority will use one of the exits and only a minority will favor the other exit. This implies that a group of humans fleeing in panic cannot be splitted equally among two escape routes. There is no use to put up two signs pointing in different directions.

On the other hand, people should be partitioned evenly among the different emergency exits of a building. One has to avoid congestion since people pushing through a narrow door can be even more dangerous than the actual emergency. This application also shows a relation between destination based routing and a *partitioning* problem.

Internet routing. Internet routing is based on *packet forwarding*. The overwhelming majority of Internet traffic is routed by *unicast forwarding*[5]. Routing decisions are solely based on the destination IP (Internet Protocol) address found in IP packets.

But with a network that big and with so many queries every second, there is a need for very fast routing algorithms. Hence, one relies on local routing strategies instead of a global optimum solution controlled by a central unit.

Routing protocols are defined in the *Internet Protocol Suite*. The two most important protocols in this suite are the *Transmission Control Protocol* and the *Internet Protocol*. Therefore, this suite is often referred as *TCP/IP*. TCP provides the service of exchanging data directly between two hosts on the same network, whereas IP handles addressing and routing messages across one or more networks.

IP again contains several different protocols for routing. In most protocols the topology of the network determines the *routing table*. One can distinguish between *Distance-vector routing protocols* and *Link-state routing protocols*. For example, *Open Shortest Path First* (OSPF) is a most widely-used protocol based on Dijkstra's algorithm.

Speaking in terms of the Internet, guideposts are *Next Hop Forwarding Tables*, which may also contain distance metric information. In this context, the guideposts are not static – an important part of these protocols is the detection of link failures and the

[5]Of course, there are also other routing strategies like multicast or broadcasting which are essential for services like WLAN (wireless local area network).

recalculation of the forwarding tables [17]. On the one hand, the network topology of the Internet changes quickly. On the other hand, exchange of information is very fast and latencies are small. Therefore, for short amounts of time, we can consider Internet routing as a destination-based routing. Furthermore, certain *virtual private network* (VPN) design problems always admit a flow-carrying tree as the optimum solution [71].

Logistics. There are also some applications for destination based routing in logistics. Many shipping companies prefer that cargo with the same destination is transported together. However, destination based routing seems to be an artificial constraint in most of these cases. It is used to keep the routing simple and it appears plausible for the dispatcher on-site. But it is probably not optimal with respect to total time or costs.

An example is described in [65]. The authors study railway cargo routing with freight trains that are recombined at large marshalling yards. Wagons with the some destination should be grouped together and they stay together until they reach their destination. However, the authors do not explicitly study the destination based routing. It is just one of many subconstraints in an integer program.

2.1.4 Intuitive description of confluent flows

Now, we are going to send flow through a network with guideposts. We will concentrate on the core of the applications presented above: the destination based routing. Therefore, we make two assumptions. First, we will only consider single commodity flows. Second, we assume that all flow units are following the guideposts.

Given sources and sinks, we choose a unique arc for outgoing flow at each node of the network. All flow particles will follow these guideposts. This additional constraint will significantly change the flow distribution. Therefore, we will introduce *confluent* flows in the next section to describe the corresponding flow pattern mathematically.

The word *confluent* originates from the Latin *confluō* (con- 'with; together' and fluō 'flow'). A flow in a network $G = (V, A, u)$ is called *confluent* if the flow uses at most *one outgoing arc* at each node. Intuitively, one can think of creeks and rivers. Thus, confluent flows can be seen as the 'most natural' flows.

Furthermore, we demand that sinks have no outgoing flow carrying arcs at all. An outgoing flow carrying arc out of a sink may imply that the flow is splitted inside the sink and only some flow is absorbed. Nevertheless, *all-or-nothing* sinks can be modeled as follows. For each sink t add a new sink t^* with the same capacity, an arc (t, t^*) and transform t to a normal node.

These restrictions have far-reaching consequences: flow carrying paths may not cross or split, and every source is assigned to exactly one sink. The total view shows the flow-carrying arcs forming an *oriented sub-forest* of the graph, pointing towards the sinks. There may still occur cycles. But since these cycles cannot consist of any sources or sinks, they do not contribute to the flow value and they can be ignored.

When the sub-forest of flow carrying arcs is fixed a priori, it is very easy to calculate the value of a maximum confluent flow on this forest by a *preflow-push-algorithm.* Starting at the sources flow is simply pushed along the arcs of the sub-forest with respect to the capacities until it reaches the sinks. Surplus flow units can be cancelled by tracing the flow back to the sources. Hence, the combined problem of finding the optimal sub-forest, i.e. putting up the guideposts, and calculating the maximum flow on this forest is the task addressed in this chapter.

From an alternative point of view, confluent flows are also linked to graph partitioning, since every source is assigned to exactly one sink. Nonsurprisingly, we will mostly face combinatorial decision problems in this chapter.

2.2 Preliminaries

After this motivation from applications, we would like to turn to a more formal description of the problem. From now on, the road network is represented by a graph-theoretical network, based on a directed graph $G = (V, A)$ without multiple arcs or loops and a capacity function u. Given a demand function d, flow units are to be sent from the sources to the sinks obeying the capacities on the arcs. Of course, we do not install guideposts in our network. Instead, we choose a unique outgoing arc at each node. None but these arcs may carry flow.

More formally, let $\mathcal{F}(V')$ be the family of sets $F \subseteq A[V']$ such that $|F \cap \delta^+(v)| \leq 1$ for all $v \in V' \subseteq V$.

Definition 2.1 (Instance). *An* instance *of a flow problem consists of a directed graph $G = (V, A)$, a capacity function $u : A \to \mathbb{N}_0$, a set of* sinks *$T \subseteq V$ with sink capacities $c : T \to \mathbb{N}_0 \cup \{\infty\}$, and supplies $d : V \to \mathbb{N}_0 \cup \{\infty\}$. The vertices $S := \{v \in V : d(v) > 0\}$ are called* sources *and we assume $S \cap T = \emptyset$.*

Definition 2.2 (Flow). *Given an instance, a* flow *is a function $x : A \to \mathbb{R}$ with $0 \leq x(a) \leq u(a)$ for all $a \in A$. A flow is called* integral *if it has only integral values. The* inflow *and* outflow *of a flow x at a given node v are* $\text{inflow}_x(v) := \sum_{a\in\delta^-(v)} x(a)$ *and* $\text{outflow}_x(v) := \sum_{a\in\delta^+(v)} x(a)$. *Let* $\text{bal}_x(v) := \text{outflow}_x(v) - \text{inflow}_x(v)$ *be the* balance *of a node. A flow x is an S-T-flow if it satisfies the* flow conservation constraint $0 \leq \text{bal}(v) \leq d(v)$ *for all $v \in V \backslash T$ and $-\text{bal}_x(t) \leq c(t)$ for all $t \in T$. The* value *of an S-T-flow is* $\text{val}(x) := -\sum_{t\in T} \text{bal}(t)$ *(as we assume $d(t) = 0$ $\forall t \in T$).*

Definition 2.3 (Confluent flow). *Given an instance and a flow x, let $F_x := \{(v, w) \in A : x(v, w) > 0\}$ be the set of flow-carrying arcs. A flow x is called* nearly confluent *if $F_x \in \mathcal{F}(V)$ and $F_x \cap \delta^+(t) = \emptyset$ for all $t \in T$. A nearly confluent flow x is called* confluent *if F_x contains no directed cycle. The* value *of a confluent S-T-flow is* $\text{val}(x) := -\sum_{t\in T} \text{bal}(t)$.

The subgraph $G' = (V, F_x)$ of flow-carrying arcs is also called *support graph* of x. For the sake of brevity, we will also simply use F_x to address this subgraph. In a confluent flow x, there is no undirected cycle in F_x. This undirected cycle would be a directed cycle as well, because it cannot contain a vertex with two outgoing arcs, which is not allowed in F_x. F_x is also an *in-forest* in this case.

Definition 2.4 (In-forest). *A directed forest with a unique sink per tree where all arcs are pointing towards a sink is called an* in-forest.

Given a graph with supplies, every support graph of a confluent flow on this graph is an in-forest, but not every in-forest is a support graph of a confluent flow. An in-forest may contain a leaf that is neither source nor sink. Thus, it cannot be reached by a confluent flow. Nevertheless, we will consider all subgraphs of G with the in-forest property as feasible set for confluent flows since unused nodes and arcs can be deleted.

An in-forest is closely related to *arborescences*, i.e. directed, rooted trees in which all arcs point away from the root. Thus, a re-orientation of the arcs in the opposite direction transforms an arborescence to an in-forest.

Furthermore, a confluent flow can never use an arc $a = (u, v)$ and the arc in the opposite direction $a' = (v, u)$ at the same time. Keeping this in mind, all definitions and results in this chapter carry over to undirected networks by considering the corresponding bi-directed graph, i.e. each edge is substituted by two opposing arcs. The positions of the sinks always imply which of the two arcs has to be used.

We can now formally define Maximum Confluent Flow and Confluent Transshipment.

Problem 2.5 (Confluent Transshipment). Input: *An instance G, u, d, S, and T.* Output: Yes *if and only if there is a confluent S-T-flow of value $\sum_{s\in S} d(s)$.*

Problem 2.6 (Maximum Confluent Flow). Input: *An instance G, u, d, S, and T.* Output: *The maximum flow value over all confluent S-T-flows.*

Clearly, an algorithm for solving Maximum Confluent Flow can solve Confluent Transshipment as well by comparing the flow value to $\sum_{s\in S} d(s)$. In contrast, we cannot use Confluent Transshipment directly, e.g. with a binary search, to compute a solution of Maximum Confluent Flow since every source and their demands have to be considered separately.

A helpful way to view confluent flows and the associated Maximum Confluent Flow and Confluent Transshipment is to break the flow computation into two parts. The first step is to determine the set of arcs that actually carry flow, in other words the in-forest F_x. Once F_x is fixed, one can use any maximum (standard) flow algorithm to compute an optimal confluent S-T-flow on F_x in a second step. (Since F_x is a forest, this subproblem is even easier than a standard flow computation.) Thus, solving any confluent flow problem essentially reduces to picking the optimum in-forest. Of course it is not clear whether this is the right point of view to handle confluent flows. However, the complexity result in Section 2.4 implies that this is at least not the worst approach.

We have already defined all capacity functions to be integral. This is not a strong restriction, since all results carry over to rational inputs after scaling and irrational inputs can be approximated. Furthermore, we assumed a loop free graph without multiple arcs. Again, this is no restriction. Loops may never contribute to the flow value and all but the one parallel arc with the highest capacity are redundant.

One important consequence is that maximum confluent S-T-flows can always be chosen integral if the input, i.e. capacities and demands, is integral, because they can be computed as maximum flows on the in-forests F_x, which are well-known to have integral solutions under this assumption.

Observation 2.7. *Maximum confluent S-T-flows can always be chosen integral, if the input is integral.*

We will now derive some more implications of the above definitions. Since we are working on a restricted arc set, the following observation is obvious.

Observation 2.8. *Given an instance, the value of* Maximum Confluent Flow *is less than or equal to the value of* Max Flow*.*

Another observation (easily understandable on trees) is that one can always drop excess units of flow in a confluent flow and trace this back to some source in the tree where the outflow can be adjusted downwards. Similar to a *preflow-push-algorithm* on a tree, the sources try to send their complete demand along the in-forest. Flow that would exceed the capacity constraint of an arc is temporarily stored in the tail of this arc. Thus, we primarily derive the amount of flow units that can reach the sinks. Afterwards, we compute the corresponding flow pattern by sending the overhead flow back to the sources. So flow conservation is not really needed, as long as no additional flow units appear out of nowhere. We will sometimes use this weaker variant of flow conservation, i.e. $\text{inflow}(v) + d(v) \geq \text{outflow}(v)\ \forall v \in V$, implicitly. For example, some algorithms in Chapter 3 will store overhead flow units.

Furthermore, we have seen that cycles do not contribute to the flow value. Some of the following algorithms will not explicitly exclude cycles, but a backtracking step from the sinks will eliminate them. The optimal flow value is already calculated without this post-processing step.

Observation 2.9. *Given an instance, the value of* Maximum Confluent Flow *is equal to the value of a maximum nearly confluent S-T-flow.*

We will finish this section with an estimate on the number of feasible in-forests for a given graph. Since Maximum Confluent Flow depends on finding the optimal in-forest, this number gives a hint on the complexity of Maximum Confluent Flow. Assume, that all nodes but the sink are sources. To determine all possible in-forests for the single sink t, we have to consider all *spanning trees* with root t in the network.

Since the sink determines the orientation of all arcs in the tree, we can use a result from undirected graphs. The number of labeled spanning trees of the complete graph K_n with labeled vertices is n^{n-2}. This famous result is called *Cayley's formula*, named after Arthur Cayley. The formula was discovered by Borchardt and proven via a determinant. A very elegant proof was given by Prüfer. He introduced the *Prüfer code* and showed that there is a bijection between the set of labeled trees with n nodes and the set of sequences over the set $\{1, \ldots, n\}$ of length $n-2$.

For short, the Prüfer code of a tree can be constructed as follows. Remove the leaf with the smallest label and set the next element of the Prüfer code to be the label of this leaf's neighbor. Stop, when only two nodes are left. Similar, a tree can be constructed out of the code. The degree of each node is equal to the number of occurrences of the label in the sequence increased by one. The proof of the Prüfer code and other proofs for Cayley's formula can be found in [3].

For multiple sinks, we are interested in the number of spanning forests, where each sink is the root of exactly one sub-tree.

Proposition 2.10 ([3]). *The number of labeled spanning forests on the complete graph K_n with k roots is kn^{n-k-1}.*

Proof. The formula can be proved very similar the Cayley's formula via a modified Prüfer code and we only present the main idea. Without loss of generality, we assume that the roots have the highest labels, i.e. label $n-k+1$ up to label n.

The sinks/roots are not connected in the desired spanning forests, but we connect them temporarily with $k-1$ arbitrary edges. Now, we are counting the number of spanning trees on the graph which are using these $k-1$ edges.

We calculate the Prüfer code as described above, but we stop when $k+1$ vertices are left. This yields n^{n-k-1} codes. It is easy to see that there exists always a leaf with a label smaller than $n-k+1$, hence none of the roots is deleted up to now and there is only one other node left. Since we have fixed the edges between the roots, the last part of the Prüfer code is redundant. We only need to know which root is the neighbor of the last node, yielding another k possibilities. □

Two other proofs for the formula in Proposition 2.10 can be found in [3]. They are parts of proofs for Cayley's formula ($k = 1$) using recursion and induction or double counting respectively. However, since the author did not find the above proof in the literature, its main idea was presented here.

Concluding, an arbitrary graph may have an exponential number of in-forests. Thus, it is not a good idea to enumerate all possible in-forests and compute a maximum flow on them if we want to solve Maximum Confluent Flow.

2.3 Related problems

Several graph theoretical problems are related to confluent flows. Despite the various applications, pure confluent flows are rarely studied in literature. In this section, we collect the main insights about confluent flows and similar problems from the literature.

2.3.1 Bottleneck paths

Given a graph $G = (V, E, u)$ with integral edge weights $u(e)$ and two special nodes s, t, the bottleneck b_P of an s-t-path P is defined as $b_P = \min_{e \in P} u(e)$. The bottleneck between s and t is defined as $\max_{P \in \mathcal{P}_{s,t}} b_P$. Each path P realizing the maximum is called a *bottleneck path*.

In other words, a bottleneck path is the thickest path between a source s and a sink t allowing to send as much flow as possible on a single path. Hence, for a single source and a single sink, the confluent flow problem is equivalent to a bottleneck path.

A bottleneck path can be found with a modified Dijkstra's algorithm. Instead of pushing the smallest label and adding lengths, one pushes the highest label first and decreases the label whenever the corresponding capacity is not sufficient. Therefore, the bottleneck path problem is solvable in polynomial time.

Another possibility is a binary search on the bottleneck capacity. One deletes all edges with a smaller capacity than the supposed capacity and checks whether s and t are still connected.

Since the bottleneck path problem is a subroutine in some algorithms for higher level problems, e.g. Max Flow, some efforts have been made to develop fast algorithms (e.g. [83]).

2.3.2 Unsplittable and k-splittable flow

An *unsplittable flow* ships the demands along a single path between a source s and a sink t. Hence, for a single commodity, a maximum unsplittable flow is equivalent to a bottleneck path or a confluent flow. Unsplittable flow becomes challenging, when multiple commodities are considered [89]. In this case, a source and a sink is given for each commodity.

However, confluent flows can be seen as taking the idea of unsplittable flows a little bit further. Still, only one path between source and sink is allowed, but additionally, paths may not cross each other. If we assume that all commodities of an unsplittable flow use the same sink, the following observation is obvious.

Observation 2.11. *For a unique sink, every confluent flow is an unsplittable flow. Thus, for a given instance, the maximum value of an unsplittable flow is an upper bound for the value of a confluent flow.*

A generalization of unsplittable flows are *k-splittable* flows. In such a flow, up to k paths between a source and a sink can be used. This problem is already hard for the case of a single source and a single sink as shown by Baier et al. [10, 11, 12]. Even worse, it is also hard to approximate a maximum k-splittable flow with arbitrary precision, i.e. there does not exist a polynomial time approximation scheme. Baier et al. present a $\frac{2}{k}$-approximation and they prove that it is $\mathcal{NP}$-hard to achieve a better performance ratio for $k = 3$. Further, the authors restrict the problem to *uniform exactly-k-splittable flows* where exactly k paths have to be used and each path has to carry the same amount of flow. In this case, a maximum s-t-flow can be found in $\mathcal{O}(km \log n)$.

2.3.3 Disjoint paths

Many variants of disjoint paths can be found in the literature. Given a graph $G = (V, E)$ and a set of pairs (s_i, t_i) of terminals, the disjoint path problems asks for pairwise disjoint paths P_i from s_i to t_i. One may consider node-disjoint or edge-disjoint paths as well as directed or undirected graphs. This decision problem is extendable to the Maximum Edge Disjoint Paths problem. Given a set of pairs of terminals, the goal is to connect as many pairs as possible with edge disjoint paths.

Consider a network with uniform arc capacities. A maximum confluent flow may have a path decomposition with many node disjoint paths. However, in confluent flows sources can be assigned to arbitrary sinks in contrast to disjoint paths where this assignment is usually fixed. Similarly, there is a relation between edge-disjoint paths and arc-confluent flows (cf. Section 2.7). Note that node disjoint paths are also edge disjoint.

Arbitrary graphs. For arbitrary directed graphs, Fortune et al. show that the Directed Vertex Disjoint Paths problem is $\mathcal{NP}$-complete even for only two pairs of terminals [62]. We will use this result for a complexity analysis of confluent flows in Section 2.4.

On undirected graphs, Robertson and Seymour [129] show that k node disjoint paths (if they exist) can be found in polynomial time for every constant k. However, when the number of terminals is not bounded by a constant, Lynch [107] proves the problem to be $\mathcal{NP}$-complete via a reduction from the satisfiability problem (SAT).

Consequently, the Maximum Edge Disjoint Paths (MEDP) problem is also hard on general graphs. Further, an approximation scheme for MEDP would contradict the hardness of the 2-Directed Vertex Disjoint Paths. In fact, this approach can be used to derive an $\mathcal{O}(\sqrt{m})$ in-approximability result. There exist a few approximation algorithms with a matching ratio. A easily comprehensible survey of MEDP related results is provided by Erlebach [54].

Planar graphs. Planar graphs are of special interest for the disjoint path problem for example due to its application in *Very-large-scale integration* (VLSI), i.e. designing integrated circuits.

Schrijver [137] proves, that the problem of finding k pairwise vertex-disjoint directed paths in a directed planar graph is solvable in polynomial time for each fixed k. Furthermore, Reed et al. [127] show that for undirected planar graphs the k disjoint paths problem can be solved even in linear time for any fixed k.

Several other restricted variants of disjoint path problems are directly related to *wire routing*. For example, Kramer and van Leeuwen [101] show that the disjoint path problem is $\mathcal{NP}$-complete on rectangular grids, i.e. finding crossing-free connections on a conductor board is difficult.

Other authors consider planar graphs where all terminals are placed on the boundary of the graph. Edge disjoint paths can quickly be found on undirected graphs with an *evenness condition* [13, 86, 148]. A graph $G = (V, E)$ with pairs (s_i, t_i) of terminals on the boundary fulfills the evenness condition if $G' = (V, E \cup \bigcup_i \{(s_i, t_i)\})$ is Eulerian.

The maximization problem MEDP is also hard on several restricted graph classes. For chain graphs, i.e. the graph consists of a single path, this problem is equivalent to finding a maximum number of pairwise disjoint intervals on the number line. This problem is solvable in linear time. For undirected trees, the problem is still in $\mathcal{P}$, but for bi-directed trees, MEDP is already $\mathcal{APX}$-hard, i.e. there does not exist a polynomial time approximation scheme [55].

2.3.4 Confluent flows with uniform arc capacities and congestion

Chen et al. [29, 30] introduce confluent flows in a different setting as considered in this thesis. Instead of obeying arc capacities, they are interested in a transshipment with a minimum node congestion, i.e. they are minimizing the maximum flow arriving at one of the sinks. Using unit demands in each node, this problem resembles a partition problem of a graph into disjoint trees of almost equal size. It is shown that there exists a *congestion gap* of $\Theta(\log n)$ compared to common flows, i.e. a significant amount of network capacity is lost due to the confluent flow condition. They derive some powerful inapproximability result and also derive an almost matching approximation algorithm. For example, they show that it is $\mathcal{NP}$-hard to approximate the congestion of an optimal confluent flow to within a factor of $\frac{1}{2}\lg k$, where k is the number of sinks in a graph G. When G admits a splittable flow with congestion at most 1, their proposed algorithm can compute a confluent flow with congestion at most $1 + \ln k$ in polynomial time.

On the one hand, one can use the Confluent Transshipment to model the decision version of their problem using sink capacities and sufficient arc capacities. On the other hand, by scaling congestion to 1, their problem can be seen as a maximum flow problem with uniform capacities. How much flow can be sent without exceeding the congestion? This problem is quite different from Maximum Confluent Flow and their inapproximability within a factor of $\frac{1}{2}\lg k - \varepsilon$ does not carry over to our problem. The crucial problem is that in the congestion case all demands are satisfied proportionally. As long as this proportion is 1, i.e. all flow must arrive at the sinks, the Maximum Confluent Flow problem behaves the same. For lower values, Maximum Confluent Flow does not enforce any proportion at all.

Furthermore, the authors of [30] study their confluent congestion minimization problem on trees and present a polynomial time algorithm for this case. We adapt some ideas of this algorithm in Section 3.1 for the Confluent Transshipment problem with heterogeneous arc capacities. Chen et al. [29] also study a demand maximization variant, where each vertex must send either nothing or its full supply, and give a 13.29-approximation for this problem. Again, this result can be seen as a maximum flow with uniform arc capacities. The authors list heterogeneous capacities among their open problems.

As a generalization, Donovan et al. [51] study d-furcated flows with node congestion, that is the graph of flow carrying arcs has at most outdegree d. Surprisingly, they present a $\left(1 + \frac{1}{d-1}\right)$-approximation for $d \geq 2$. Additionally, the congestion gap is bounded by the same ratio. Thus, increasing the maximum outdegree from one to two almost eliminates the congestion gap of $\Theta(\log n)$ of confluent flows.

2.3.5 Mixed integer programming and confluent flows

To the best of our knowledge, there are no combinatorial approaches to solve confluent flows with heterogeneous arc capacities on non-trivial graph classes up to now. Nevertheless, confluent flows appear in some applications and most authors addressed them by mixed integer programming.

As mentioned in the introduction, Fügenschuh et al. [65] integrate confluent flow constraints into their mixed integer programming formulation for a railway routing problem. Due to the many other integral constraints in their application, integer programming is recommended.

A more detailed analysis of mixed integer formulations for confluent flows is stated by Achterberg [125]. Under the name of *arboricity flows*, he studies an improved linear programming relaxation and derives classes of cutting planes. Under certain conditions, these inequalities can describe facets of the flow polytope.

2.3.6 Confluent flows over time on trees

In a series of papers Fujishige et al. [108, 109, 110, 111] consider confluent *flows over time* (cf. Section 4.2.4) with multiple sinks but restricted to trees. The routing decisions are reduced to a sink location problem in this case. Further, the authors study evacuation as an application.

In Section 2.4, we will see that confluent flows are hard to compute even on trees. Somewhat surprising, this time-dependent problem can be solved in polynomial time even though the related static MAXIMUM CONFLUENT FLOW problem cannot. One key difference is that arc capacities in flows over time limit the flow rate and not the total flow on an arc as in the static version. Therefore, additional flow can be sent over each link in the network at the expense of more time. Thus, more flow can always be achieved with patience, whereas a static flow problem simply becomes infeasible if the given capacities are insufficient. Finally, Fujishige et al. present an $\mathcal{O}(n \log n)$ algorithm for their sink location problem in dynamic tree networks.

2.3.7 Spanning trees, minimum cost flows and setup costs

Given a connected, undirected graph G, a *spanning tree* of G is a subgraph that is a tree and contains all vertices of G. Given edge weights, one may ask for spanning tree with minimum or maximum overall weight. There exist several polynomial time algorithms for find minimum spanning trees, e.g. the algorithm of Prim and Kruskal's algorithm. Both are greedy algorithms.

The *k-minimum/maximum spanning tree*, which is the tree that spans some subset of k vertices in G with minimum/maximum weight, seems to be closely related to the problem above. In contrast, this problem is $\mathcal{NP}$-hard [9]. Another related problem is the *Steiner tree problem* on graphs. Given a subset $S \subseteq V$ a Steiner tree is a tree in G that spans all vertices of S. It may contain vertices of $V \backslash S$. Again, one may ask for minimum Steiner trees. The corresponding decision problem is one of Karp's 21 $\mathcal{NP}$-complete problems.

As seen above, there are several problems where a tree or a subtree of a graph is searched for. For confluent flows, we are also looking for a subtree with high capacities on the arcs. However, the above problems do not consider any additive behavior of a confluent flow towards the sink. For confluent flows the highest capacity is needed next to the sink. Hence, on the one hand, the above problems seem to be rather loosely related to confluent flows.

On the other hand, consider minimum cost flows as introduced in Problem 1.3. A some what challenging variation of minimum cost flows are flows with setup costs. Hereby, a certain setup cost or toll has to be paid if an arc is going to be used. This toll is independent of the flow value on this arc. Now, the objective is to ship a certain amount of flow at minimum cost. For infinite capacities, a minimum cost Steiner tree is the optimal solution for the flow carrying arcs. Thus, the minimum cost flow is confluent.

2.4 Complexity

In Section 2.2, we gave an estimation on the number of feasible in-forests for confluent flows. However, a high number of possible routings does not imply the hardness of confluent flows. Thus, we will now have a closer look at the complexity of confluent flows.

First, let us fix the decision version of Maximum Confluent Flow to discuss its hardness: "Given an instance G, u, d, S, T and c, is there a confluent S-T-flow with value at least f?".

We start with a result for general directed graphs. In Section 2.3 we presented the Directed Vertex Disjoint Path Problem. We will use its relationship to confluent flows for a reduction. One main difficulty of disjoint paths is the assignment of the sources to the corresponding sink. We will use arc capacities to model this assignment with confluent flows. The setting is also displayed in Figure 2.

Theorem 2.12. *Approximating* Maximum Confluent Flow *on general directed graphs within a factor of* $2/3 + \varepsilon$ *is* $\mathcal{NP}$*-hard.*

Proof. Consider the k-Directed Vertex Disjoint Path Problem: Given a directed graph G with k terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$, are there k node-disjoint paths that connect the corresponding terminals? Even for $k = 2$, this problem is strongly $\mathcal{NP}$-complete [62].

This can be transformed to a Maximum Confluent Flow on the same graph with arc capacities $u \equiv 2$. Each source s_i, $i \in \{1, 2\}$, has supply $d(s_i) := i$ and each sink t_i has capacity $c(t_i) := i$. The maximum flow value is 3 if and only if the two disjoint paths exist.

When using the disjoint paths, a confluent flow value of 3 is obvious. Otherwise, if there are no disjoint paths, one of the following is true:

- The paths of the confluent flow meet, one sink cannot be used at all.
- The terminals are mismatched, i.e. s_1 is connected to t_2 and s_2 is connected to t_1.
- One pair of terminals is not connected at all.

However, the optimum value of a single commodity confluent flow is at most 2. Hence, any confluent flow with a flow value of $2 + \epsilon$ would imply that two disjoint paths have been found. Thus, it is $\mathcal{NP}$-hard to approximate Maximum Confluent Flow on general directed graphs within a factor of $2/3 + \varepsilon$. □

Note that the capacities are chosen in a well-considered way, that is all of the three cases above imply the same value of a maximum confluent flow. Furthermore, the value of a maximum confluent flow on a fixed in-forest can easily be computed. Thus, we have a polynomial-time checkable certificate for the Yes-answer of the decision version

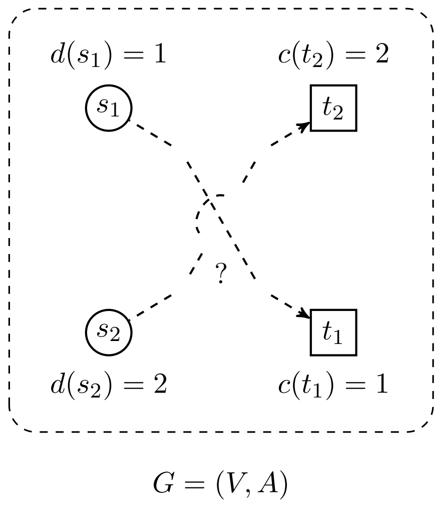

Figure 2: Graph with two pairs of terminals, all arc capacities are 2. If we can find node disjoint paths connecting s_1 with t_1 and s_2 with t_2, then we can use these paths to ship 3 units of flow confluently. If no such paths exist, then we may only send 2 flow units.

of the confluent flow problem. A non-deterministic algorithm just guesses the right in-forest, everything else can be computed in polynomial time. Hence, $\mathcal{NP}$-hardness results for confluent flows presented in this thesis imply $\mathcal{NP}$-completeness of the considered problems.

With our practical application in mind, the $\mathcal{NP}$-completeness of the Maximum Confluent Flow is discouraging but not surprising at all. We will now consider confluent flows on several graph classes. We will start with the most restrictive graph class for confluent flows. If G is a tree, it should be easy to determine the optimal in-forest. Hence, the following result is surprising.

Theorem 2.13. Maximum Confluent Flow *is $\mathcal{NP}$-hard if G is a directed tree and all demands and supplies are infinite.*

Proof. We reduce the weakly $\mathcal{NP}$-complete Partition problem [66] to the decision version of Maximum Confluent Flow. An instance of Partition consists of n positive integers $d_1, \dots, d_n$ with $D := \sum_{i=1}^{n} d_i$ and the question is, whether there is a subset $P \subseteq \{1, \dots, n\}$ such that $\sum_{i \in P} d_i = D/2$.

To model this as a Maximum Confluent Flow, consider a graph with vertex set $V = T' \dot{\cup} S \dot{\cup} \{v, t^*\}$ where $S = \{s_1, \dots, s_n\}$ is the set of sources, $T' = \{t_1, \dots, t_n\}$ and t^* are sinks, and v is a normal vertex. The arc set is $E = \{(s_i, t_i), (s_i, v) : i = 1, \dots, n\} \cup \{(v, t^*)\}$, that is the sources can either go to "their own" sink or through v to a common sink t^*. The arc capacities are $u(s_i, t_i) = d_i$, $u(s_i, v_i) = 2d_i$, and $u(v, t^*) = D$. All source and sink capacities are infinite. See Figure 3 for the construction.

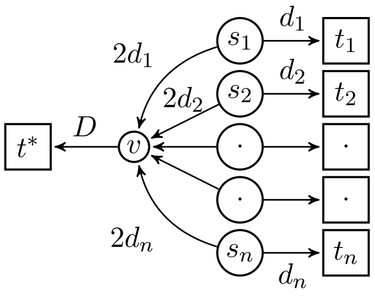

Figure 3: This tree models a Partition instance using n sources $\{s_i\}$, and $D = \sum_{i=1}^{n} d_i$. Labels at arcs refer to capacities. The optimum confluent flow value is $\frac{3}{2}\sum_{i=1}^{n} d_i$ if and only if partitioning $\{2d_i : 1 \leq i \leq n\}$ into two sets of equal sum is possible.

Seen locally, a source s_i may send twice as much flow to t^* than to t_i. From a global perspective, flow sent to t^* could be wasted if the capacity of (v, t^*) is already utilized.

We claim that the solution to Maximum Confluent Flow is at least $f := \frac{3}{2}D$ if and only if the Partition instance is a Yes-instance.

If there is a solution P to the Partition instance, we direct the flow from each s_i with $i \in P$ to v, and s_i with $i \notin P$ to t_i. Of course, v should direct its flow to t^*. The flow value of this solution is d_i for $i \notin P$ and $2d_i$ for $i \in P$, noting that $\sum_{i \in P} 2d_i = D = u(v, t^*)$. This yields a flow of value $\frac{3}{2}D$ as claimed.

Suppose a flow x with value at least $\frac{3}{2}D$ is given. Let P contain those indices i such that s_i tries to send flow to t^* and let $z := \sum_{i \in P} d_i$. With respect to the capacities, $\min\{2z, D\}$ flow units can reach t^*. Consequently, the remaining sources s_i, $i \notin P$, can send a total amount of $D - z$ flow units to T'. If $z > D/2$, then the value of the flow is at most $D + (D - z) < \frac{3}{2}D$, a contradiction. If $z < D/2$, then the flow value is at most $2z + (D - z) < \frac{3}{2}D$, again a contradiction. So $z = \sum_{i \in P} d_i = D/2$ must hold, and the Partition instance is a Yes-instance. $\square$

On the one hand, it follows that, although confluent flows are closely linked to trees, the Maximum Confluent Flow problem is hard on trees. This result implies hardness for several other graph classes, even when using only a single sink.

Corollary 2.14. Maximum Confluent Flow *with a single sink is $\mathcal{NP}$-hard on planar graphs with treewidth 2 (cf. Section 3.2 for the definition of treewidth).*

Proof. Note that in the proof of Theorem 2.13 we can identify all of $T = T' \cup \{t^*\}$ with t^*. This yields a planar graph that has treewidth 2 with only one sink. Another modified instance with treewidth 2 is presented in Figure 4. $\square$

On the other hand, the Maximum Confluent Flow is very easy on a tree with a single sink, since the routing is completely fixed in this case. Thus, the border of easy

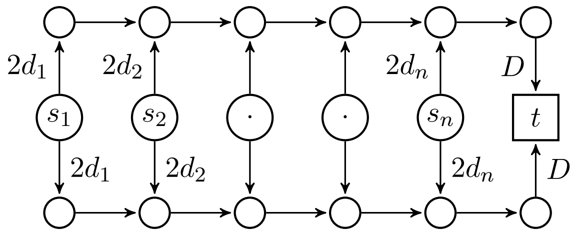

Figure 4: The modified PARTITION instance with a single sink. This instance implies that MAXIMUM CONFLUENT FLOW with one sink is at least weakly $\mathcal{NP}$-hard on $3 \times n$ grids, series-parallel graphs (or graphs with treewidth 2) and 2-outerplanar graphs, all of which are planar in particular.

and hard problems is somewhere in between. We will investigate special graph classes and restrictions to the number of terminals in Chapter 3.

2.5 A mixed integer programming formulation

In Section 1.4.1, we introduced a linear program for the MAX FLOW problem. We will extend this LP to confluent flows, now. Obviously, we will need some integral variables to restrict the number of outgoing arcs. Given an instance $G = (V, A)$, u, d, S, t and $c(t) = \infty$, we formulate the following mixed integer program (MIP).

$$
\begin{aligned}
\max \quad & \sum_{e=(v,t)} x(e) & & (5)\\
\text{s. t.} \quad & u(e)b(e) - x(e) \geq 0 & \forall e \in A \qquad & (6)\\
& 0 \leq \sum_{e=(u,v)} x(e) - \sum_{e=(v,w)} x(e) \leq d(v) & \forall v \in V \backslash \{t\} \qquad & (7)\\
& \sum_{e \in \delta^+(v)} b(e) \leq 1 - \delta_{vt} & \forall v \in V \qquad & (8)\\
& x(e) \geq 0 & \forall e \in A \qquad & (9)\\
& b(e) \in \{0, 1\} & \forall e \in A \qquad & (10)
\end{aligned}
$$

The objective (5) is the sum over the incoming flow into the sink t. In constraint (6), the capacity of each arc is multiplied with the corresponding binary decision variable. Thus, the binary variable switches the arc *on* or *off*. Flow conservation and demands are handled in constraint (7). The next constraint limits the number of flow-carrying outgoing arcs of each node. For short, we use Kronecker's delta δ_{vt} to forbid outgoing arcs out of the sink, i.e. $\delta_{vt} = 1$ if $v = t$ and 0 otherwise.

Observation 2.15. *Solving the mixed integer program (5)–(10) yields a maximum confluent S-T-flow for the underlying network.*

This mixed integer formulation can also be extended to multicommodity confluent S-T-flows.

$$
\begin{aligned}
\max \quad & \sum_{i\in\mathcal{I}} \sum_{e=(v,t_i)} x_i(e) && && (11)\\
\text{s.t.} \quad & u(e)b_i(e) - x_i(e) \geq 0 && \forall e \in A\ \forall i \in \mathcal{I} && (12)\\
& u(e) - \sum_{i\in\mathcal{I}} x_i(e) \geq 0 && \forall e \in A && (13)\\
& 0 \leq \sum_{e=(u,v)} x_i(e) - \sum_{e=(v,w)} x_i(e) \leq d_i(v) && \forall v \in V\backslash\{t_i\}\ \forall i \in \mathcal{I} && (14)\\
& \sum_{e\in\delta^+(v)} b_i(e) \leq 1 - \delta_{vt_i} && \forall v \in V\ \forall i \in \mathcal{I} && (15)\\
& x_i(e) \geq 0 && \forall e \in A\ \forall i \in \mathcal{I} && (16)\\
& b_i(e) \in \{0,1\} && \forall e \in A\ \forall i \in \mathcal{I} && (17)
\end{aligned}
$$

Assuming that each commodity has its own destination, we use separate flow functions for each commodity. There are two capacity contraints, now. The first constraint (12) models the binary switches for each commodity. The second constraint (13) guarantees that the capacity of an arc is shared among all commodities. All other constraints apply for each commodity.

But there is also another important observation. For many related flow problems, e.g. standard Max Flow or k-splittable flow, one can find a practicable path based formulation (cf. Section 1.4.1). Such a formulation admits a column generation approach. The columns correspond to shortest paths, calculated iteratively to improve the flow value. A similar path based approach for confluent flows does not seem to exist, because a crucial problem occurs. Consider two paths in the MIP formulation. We need an additional constraint that permits flow on both paths only if they do not cross each other. Such constraints can be formulated, but there seems to be no way around binary decision variables for each path. Much more binary variables are needed than in the arc based formulation. Furthermore, thinking of column generation for solving the MIP, adding a new path to the MIP may conflict with all other paths found so far.

Consequently, one may think of a tree based mixed integer formulation, now. Similarly, listing all in-trees is not a good idea due to their high number. In contrast to paths, only one in-tree is used for a confluent flow with a single sink. Thus, in a column generation approach, only one column can be used. An algorithm that computes a new column would actually compute a better in-tree. Such an algorithm would nearly solve the Maximum Confluent Flow problem by itself.

2.6 A max-flow min-cut theorem

The *Maximum Flow Minimum Cut theorem* (cf. Section 1.2.2) is one of the most important results in network flow theory. In this section, we will have a closer look at 'confluent' cuts and we will try to derive a result similar to Max Flow-Min Cut.

In this section, we assume that all sources and sinks have infinite demand or capacity respectively. That is not a strong restriction. If a source s has finite demand $d(s) > 0$, we introduce a new source $\bar{s}$ and an arc $a = (\bar{s}, s)$ with capacity $u(a) = d(s)$. Finally, we set $d(s) = 0$ and $d(\bar{s}) = \infty$. Obviously, the maximum flow value remains unchanged. However, we may consider a in the computation of a cut. One can use a similar construction for the sinks or one may add a supersink.

Non-surprisingly, the capacity of a standard cut is an upper bound on the value of the Maximum Confluent Flow. But a standard cut will in most cases overestimate a confluent S-T-flow, since only one outgoing arc can be used at each node. That is, the outflow of each node is obviously bounded by the highest capacity of the outgoing arcs. But there is no guarantee that an arc with the highest capacity is used at all. Thus, we need a more sophisticated cut definition. On the other hand, confluent S-T-flows are hard to compute. Hence, an associated cut – which marks the critical arcs – is also not trivial to derive.

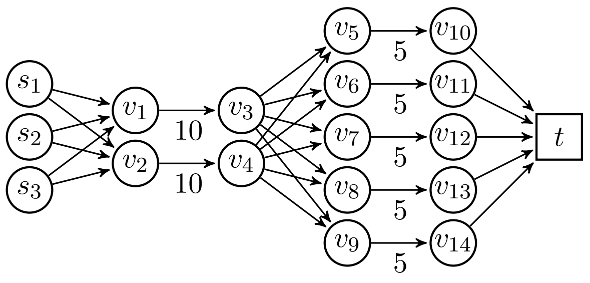

Figure 5: What is an appropriate cut definition for confluent flows? The arcs are labeled with their capacities. Unlabeled arcs have infinite capacity. If we consider the pair of parallel arcs, 20 flow units may reach the sink. This would be a minimum cut for standard flows. The five parallel arcs itself can carry 25 flow units. However, the maximum value of a confluent flow is 10.

The evidential quality of a standard cut is limited since we have no information whether flow was merged before the cut or not. Figure 5 suggests the following definition.

Definition 2.16 (Confluent cut). *A* confluent cut *consists of two node sets C_1, C_2 with $C_1 \subseteq C_2 \subseteq V$ such that $S \subseteq C_1$ and $T \subseteq V \backslash C_2$.*

A confluent cut defines two consecutive frontiers between the set of sources and the set of sinks. Note that $C_1 \subseteq C_2$, i.e. flow that was already merged inside C_1 has to leave C_2 on a single arc, too. If x flow units are going to be sent out of C_1 on a single arc, then we need an outgoing arc of C_2 which provides at least the same capacity. Unaffected thereof, flow can be merged after it left C_1 and before it leaves C_2.

This behavior resembles *bin packing*. Here, a set of *items* with weights w_i should be assigned to bins of size b_j, such that the total weight of items assigned to a bin does not exceed the size of this bin. Items must not be splitted, but several items may be assigned to a single bin. With bin packing in mind, the capacity of a confluent cut is derived as follows.

Definition 2.17 (Capacity of a confluent cut). *For $i = 1, 2$ let A_i be the set of outgoing arcs of C_i, i.e. $A_i = \{a \in A : tail(a) \in C_i, head(a) \notin C_i\}$. Let A'_i be an arbitrary subset of A_i such that each node in C_i is incident to at most one arc in A'_i.*

Now consider a bin packing of items of size $u(a)$ $\forall a \in A'_1$ into bins of size $u(a)$ $\forall a \in A'_2$. The volume of a bin may be exceeded, but at most the size of the bin is accounted. That is, the value of such a packing is the filled volume of the bins.

The capacity of a confluent cut *is the maximum value of such a bin packing.*

Since the capacity of a confluent cut is defined as a maximum, one has to choose the arcs with the largest capacities in A_i to obtain optimal subsets A'_i.

Applying this definition to the graph in Figure 5, we choose $C_1 = \{s_1, s_2, s_3, v_1, v_2\}$ and $C_2 = \{s_1, s_2, s_3, v_1, \ldots, v_9\}$. We have to assign two packets of size 10 to five bins of size 5. Hence, the capacity of this confluent cut is 10.

Theorem 2.18. *Given an instance, the capacity of a confluent cut is greater than or equal to the value of a confluent S-T-flow.*

Proof. Confluent flow may leave a node only on one outgoing arc. On the one hand, at most $\sum_{a \in A'_i} u(a)$ flow units may leave C_i for $i = 1, 2$. Flow that leaves C_1 on a single arc is not allowed to split. Hence, this flow has to leave C_2 on a single arc, too. This is modeled by the bin packing.

On the other hand, given a confluent S-T-flow, we may restrict A_i to the in-forest of flow carrying arcs. This yields another bin packing instance. The number of packets and bins is less than or equal to the number of packets and bins in the original instance. Since the in-forest may not take the arcs with the largest capacities packets and bins may be even smaller. Thus, the maximum value of this new bin packing is less than or equal to the maximum value of the original packing.

If we consider the flow values $f(a)$ instead of the capacities $u(a)$ for defining the bin packing, this yields a bin packing instance with even smaller value. However, this instance has a solution where all bins are filled exactly due to flow conservation and the underlying tree structure. Obviously, this value is equal to the value of the confluent flow. $\square$

Unfortunately, the minimum value of a confluent cut is not equal to the maximum value of a confluent S-T-flow in general. We may assign items to arbitrary bins. In contrast, node disjoint paths between the corresponding arcs may not exist in the network, i.e. this assignment cannot be realized in a confluent flow. The example in Figure 6 shows that the gap between flow value and cut capacity can be at least $\frac{3}{2}$.

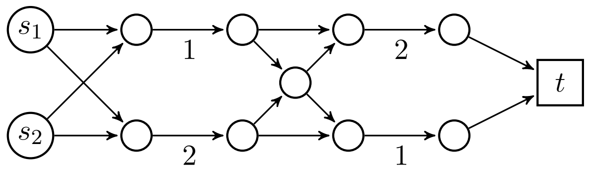

Figure 6: In the presented graph, the minimum value of a confluent cut is 3. However, a flow of value 3 cannot be established without crossing or splitting paths. Thus, the value of a maximum confluent S-T-flow is 2.

Nevertheless, equality holds in some cases.

Proposition 2.19. *For a single source and a single sink, the value of maximum confluent S-T-flow is equal to the value of a minimum confluent cut.*

Proof. For a single source s and a single sink t, a maximum confluent S-T-flow becomes a bottleneck path problem (cf. Section 2.3). Given an instance, we choose $C_1 = \{s\}$. Thus, A'_1 contains a single arc. W.l.o.g. this arc has infinite capacity. Let C_2 be an arbitrary subset of $V \backslash \{t\}$. The bin packing identifies the arc with the highest capacity in A'_2, since only one packet (of infinite size) can be assigned to a bin. Thus, the confluent cut definition is equal to finding the classical cut where the maximum arc capacity is minimum.

The identified arc is the bottleneck arc. Since it is the arc with the highest capacity in the cut, deleting all arcs with this capacity or any lower capacity disconnects source and sink. Thus, the capacity of this arc is equal to the capacity of the bottleneck path and, consequently, it is equal to the value of a confluent flow in this instance. $\square$

Proposition 2.20. *For uniform capacities and uniform demands, the value of maximum confluent S-T-flow is equal to the value of a minimum confluent cut.*

Proof. For uniform capacities and uniform demands, the problem is reduced to node disjoint paths between S and T. There exists an integral solution, thus we may use each flow carrying path P with $f(P) = 1$. Consequently, no flow carrying paths meet, since they cannot be merged. Hence, there exists an optimal confluent flow which uses node disjoint paths. The maximum number of such paths between S and T is equal to the minimum number of nodes in a separator of S and T (Menger's Theorem).

Choose arbitrary sets C_1 and C_2 with $S \subseteq C_1 \subseteq C_2 \subseteq V \setminus T$. $\bar{C}_i = \{v \in C_i : \exists (v,u) \in A, u \notin C_i\}$ defines a node separator of S and T. For $u \equiv 1$, the capacity of a confluent cut simple counts the minimum number of nodes in both separators $\bar{C}_i$, since only one outgoing arc per node is considered. Obviously, the minimum capacity of a confluent cut is equal to the minimum number of nodes in a separator. Thus, Menger's theorem proves the claim. $\square$

The following proposition limits the gap between the minimum capacity of a confluent cut and the maximum value of a confluent S-T-flow.

Proposition 2.21. *Given an instance with k sources, the value of a minimum confluent cut is at most k times the value of a maximum confluent S-T-flow.*

Proof. Let $s_1, \dots, s_k$ be the k sources. We insert a supersink t that is reachable from all sinks via an arc with infinite capacity. We consider the k bottleneck paths from each source to the sink t. Obviously, we cannot send more flow than the sum over the capacities of these k paths, but we can send at least flow on the path with the highest capacity. Let P be this path with the maximum bottleneck u_P.

We choose $C_1 = \{s_i : i = 1, \dots, k\}$, i.e. A'_1 contains at most k arcs. C_2 is chosen as in the proof of Proposition 2.19 to be the bottleneck cut for P. Thus, A'_2 consists of arcs with capacity at most u_P. Therefore, the value of the bin packing can reach at most ku_P.

That is, the maximum confluent flow value is at least u_P and the minimum confluent cut capacity is at most ku_P. □

Unfortunately, the presented confluent cut definition is not very strong. On the one hand, it is difficult to keep track of crossing paths when considering only a small set of arcs. On the other hand, confluent flows are hard to compute. Thus, we should not hope for an easy definition of a confluent cut that is easy to compute at all. Closing this section, we will suggest some ideas for an improvement of confluent cuts.

For arbitrary graphs, the bin packing idea can be applied consecutively. Instead of only two sets C_1 and C_2, we can consider a *cascade* of node sets C_i, $i = 1, \dots, N$, with $S \subseteq C_i \subset C_{i+1} \subseteq V \backslash T \; \forall i \in \{1, \dots, N\}$. The filled bins corresponding to C_i are considered as items, i.e. the filled volume is accounted as item size, and these items are assigned to bins representing the outgoing arcs of C_{i+1}. Obviously, the bin packing instances are not independent from each other. Thus, it is already difficult to compute an optimal consecutive packing for a given cascade. Nevertheless, an appropriate cascade may help to reduce the capacity of a confluent cut significantly.

For planar graphs, we can use an embedding of the graph to define an *ordered bin packing*. Items, i.e. arcs corresponding to items, cannot be assigned to arbitrary bins, since an embedding would imply crossing paths. An example is presented in Figure 7. Here, the embedding implies an ordering of items and bins. If we assign an item to a bin, no item with a higher index may be assigned to a bin with a lower index. However, we have to be very careful, since it is not clear whether another embedding of the same graph may yield a different capacity of the confluent cut. Furthermore, the example in Figure 7 also demonstrates that the gap is not closed completely.

Finally, the presented confluent cuts provide quite useful bounds in practice. The estimation in Proposition 2.21 seems to be poor and can probably be improved. Based on Theorem 2.12, one may conjecture that the ratio between a minimum confluent cut and a maximum confluent flow in a network with two sources is at most 1.5.

Summarizing, the definition of a confluent cut and its implications can certainly be improved. Thus, we list confluent cuts as one of the most interesting open problems related to confluent flows in the outlook in Section 3.5.

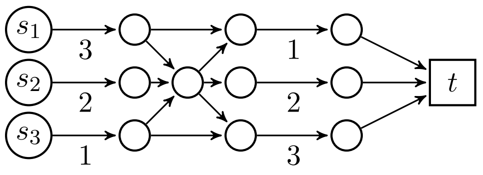

Figure 7: The embedding implies the sequence $(3, 2, 1)$ of item weights and the sequence $(1, 2, 3)$ of bin sizes. If we assign the ith item to the jth bin, the mapping $i \mapsto j$ has to be non-decreasing, since paths must not cross. Here, the maximum confluent flow is 4 (cf. Section 3.4). The value of a confluent cut is 6. Considering planarity, the upper bound on the flow value is reduced the 5 $(1 \mapsto 2,\ 2 \mapsto 3,\ 3 \mapsto 3)$, but the gap is not closed.

2.7 Arc-confluent flows

With the above definitions the routing decision is made in the nodes of the network. Flow units that meet in a node are merged. In contrast, one may also consider a variant of the problem where only flow that meets on a common arc has to stay together. Speaking in the terms of guideposts, the guidepost is located at the road before arriving at the intersection. We will call such a flow *arc-confluent*.

2.7.1 Definition of arc-confluent flow

To simplify matters, we assume that each source has exactly one outgoing arc. This is no restriction. For each source $s \in S$ in the original network, we introduce a new source $\bar{s}$ and an arc $a = (\bar{s}, s)$ with capacity $u(a) = d(s)$. Source demands are adjusted to $d(s) = 0$ and $d(\bar{s}) = \infty$, i.e. s is no source anymore (c.f. Section 2.6).

Definition 2.22 (Nearly arc-confluent flow). *Given an instance and an S-T-flow x, this S-T-flow x is called* nearly arc-confluent *if there exists a mapping $M_v : \delta^-(v) \to \delta^+(v)$ for each node $v \in V \backslash (S \cup T)$ such that $\sum_{a \in M_v^{-1}(\bar{a})} x(a) = x(\bar{a})\ \forall \bar{a} \in \delta^+(v)$, $|\delta^+(s)| = 1\ \forall s \in S$, and $\delta^+(t) = \emptyset\ \forall t \in T$.*

A nearly arc-confluent S-T-flow may use several outgoing arcs at each node. Note that the above definition also ensures flow conservation. Additionally, the graph induced by the flow carrying arcs may contain several directed cycles. Neither these cycles can simply be ignored nor can they be cancelled like cycles in standard flows. Figure 8 shows an example that cancelling cycles may violate the nearly arc-confluent condition.

The following definitions deal more carefully with forbidden cycles. There are three kinds of cycles. In the first case, there exists a mapping, where flow units actually circulate, i.e. no flow particles enter or leave this cycle. The flow on this cycle can be reduced by an arbitrary value. In the second case, there exists a mapping which implies a path decomposition of the flow, where a path passes a node twice. This loop can be deleted. Both cases are displayed in Figure 9. Before we look at cycles of the third

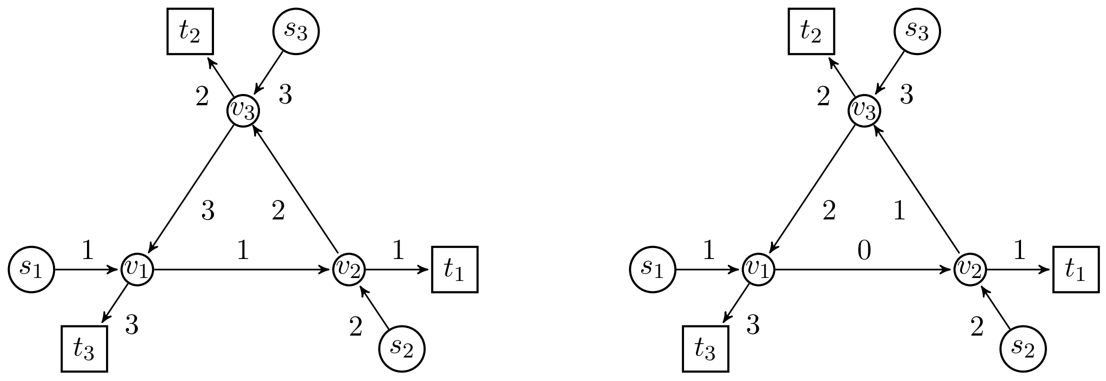

Figure 8: Cycles in nearly arc-confluent S-T-flows cannot simply be cancelled. The labels at the arcs correspond to the flow values. Obviously, the flow on the left side is nearly arc-confluent. We may try to cancel the cycle by pushing one unit of flow backwards. This yields the flow on the right side. This flow is not nearly arc-confluent, because there is a node with one incoming and two outgoing flow carrying arcs.

kind, we exclude arc-confluent flows which contain cycles of category 1 and 2 from our considerations with help of Definition 2.23.

Definition 2.23 (Weakly arc-confluent flow). *Given an instance and a nearly arc-confluent S-T-flow x, this S-T-flow x is called* weakly arc-confluent *if for all feasible mappings $M_v : \delta^-(v) \to \delta^+(v)$ as in Definition 2.22 there does not exist a cycle $(a_1, \dots, a_k)$ such that $a_{i+1} = M_{head(a_i)}(a_i)\ \forall i \in \{1, \dots, k-1\}$.*

The support graph of a weakly confluent flow may still contain a directed cycle of the third kind. Flow on these cycles cannot be reduced by simply turning it back. A re-routing of several flow carrying paths is necessary to delete these cycles. An example of this third case is shown in Figure 10.

The following definition also excludes the third kind of cycles.

Definition 2.24 (Arc-confluent flow). *Given an instance and a weakly arc-confluent S-T-flow x, this S-T-flow x is called* arc-confluent *if for any value $r > 0$ there exists no cycle $C = (a_1, \dots, a_k)$ such that the flow $\bar{x} := \begin{cases} \bar{x}(a) = x(a) - r & : \quad a \in C \\ \bar{x}(a) = x(a) & : \quad a \notin C \end{cases}$ is weakly arc-confluent.*

Arc-confluent flows can be seen as the less restrictive variant of confluent flows. Flow on a common arc also uses a common node which implies Observation 2.25.

Observation 2.25. *Every confluent S-T-flow is also arc-confluent.*

Thus, confluent flows and arc-confluent flows have a similar relation as arc disjoint paths and node disjoint paths. Observation 2.25 leads directly to the following observation.

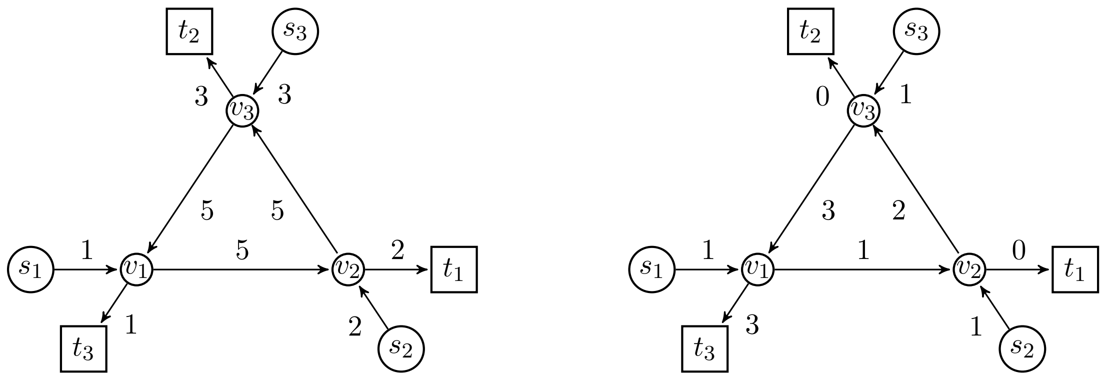

Figure 9: A nearly arc-confluent S-T-flow can contain several kinds of cycles. On the left side, there is a circulation that can be deleted. On the right side, a flow particle from s_1 is send to t_3 via v_1, v_2, v_3, and v_1 again. Thus, flow on the inner cycle can be reduced by exactly one flow unit by sending flow from (s_1, v_1) directly to (v_1, t_3).

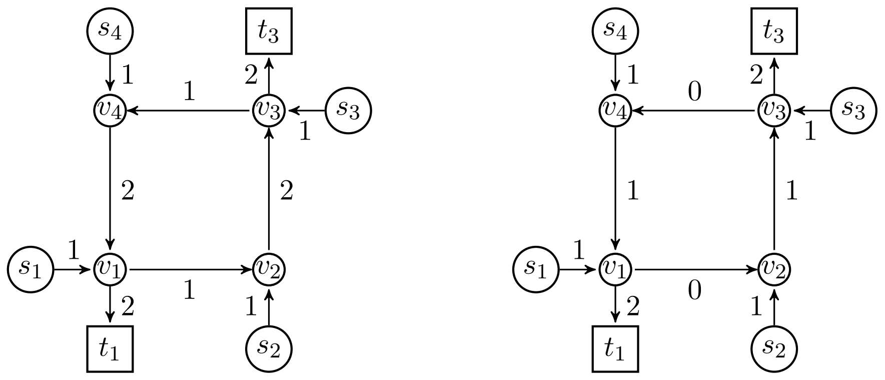

Figure 10: The flow in the left network is weakly arc-confluent. In contrast to the flows in Figure 9, the inner cycle is build up by various paths. Nevertheless, flow on this cycle can be reduced by several re-routings as demonstrated on the right side.

Observation 2.26. *Given an instance, the value of a maximum arc-confluent S-T-flow is greater than or equal to the value of a maximum confluent S-T-flow.*

2.7.2 Complexity results for arc-confluent flows

In the previous section, we introduced *arc-confluent flows* with fewer restrictions on the routing than confluent flows. We will now present two (polynomial-time) reductions to transform an arc-confluent flow problem on a directed graph into a confluent flow problem and vice versa. Thus, the main complexity result from confluent flows carries over to arc-confluent flows.

Theorem 2.27. *It is $\mathcal{NP}$-complete to decide whether an instance G, u, d, S, T and c allows an arc-confluent flow of value f.*

Proof. $\Rightarrow$: Assume, we have an instance G, u, d, S, T and c and we want to compute a confluent flow. We construct a graph G' as follows. For each vertex $v \in V$ we add two vertices v_{in} and v_{out} to G', demands are transmitted to v_{in}. These vertices are connected via arcs (v_{in}, v_{out}) with infinite capacity. For each arc $a = (v, u)$ in G we add the arc (v_{out}, u_{in}) with the same capacity to G'. In other words, each node is split into an incoming and an outgoing node. Obviously, an arc-confluent flow on G' also solves the original problem on G. All incoming flow of a node is forced on a common arc, where the arc-confluent flow condition applies. Now, only one outgoing arc of the outgoing node can be used. The construction is depicted in Figure 11.

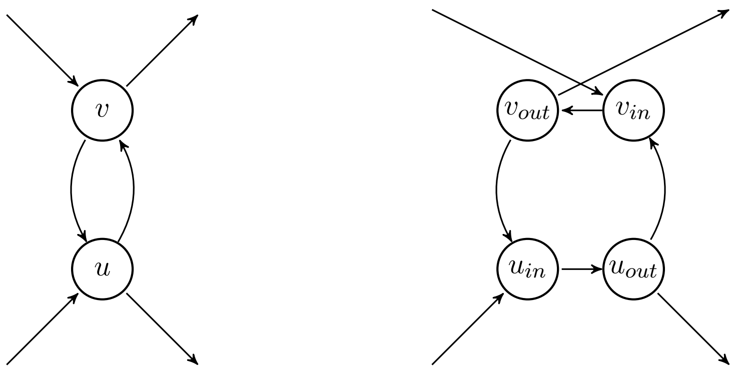

Figure 11: Transformation of an instance of confluent flow into a corresponding arc-confluent flow problem. We observed that a confluent flow may never use an arc and its backward arc at the same time, because this would induce a cycle. This carries over to arc-confluent flows on graphs obtained with the construction in the proof of Theorem 2.27. The nodes for incoming flow have only one outgoing arc and the corresponding cycle consists of four arcs.

$\Leftarrow$: Now, assume, we want to compute an arc-confluent flow for an instance G, u, d, S, T and c. Arc-confluent flow may bypass in a node. This can be modeled by a confluent flow on the *line graph* of G. For our purpose, the line graph G' of G is constructed as follows. For each arc a in G, add a node v_a to G'. For each pair (a_1, a_2) of arcs in G with $\text{head}(a_1) = \text{tail}(a_2)$ the arc (v_{a_1}, v_{a_2}) is added to G'. For each arc a in G the capacity of a is assigned to all outgoing arcs of v_a in G'. Since only one outgoing arc can be used in the confluent flow, we have not to care about capacity sharing. Source and sink demands cannot directly be added to G', since the original terminal is splitted into its several route choice possibilities, so to say. Instead, we add a new node with the same demand to G' and connect it to all nodes which correspond to the outgoing arcs or incoming arcs of the original terminal respectively. Likewise as above, a confluent flow on G' also solves the original problem on G. All arc-confluent flow which would have met on a is merged in v_a now.

Obviously, the transformation of the problem can be done in polynomial time in both cases. Reconstructing the solution of the original problem is also straightforward. $\square$

Thus, confluent flow and arc-confluent flow are very similar. Note that the reduction is not applicable for special graph classes, since G' may not belong to the same class anymore. Furthermore, this transformation applies only to directed graphs, but undirected graphs can be processed by making them bi-directed. From now on, we will mainly consider confluent flows on directed graphs.

3 Algorithms for Confluent Flows

Up to now, we have seen that MAXIMUM CONFLUENT FLOW is a challenging problem on arbitrary graphs. This discouraging (complexity) result motivates to study confluent flows on special graph classes. However, even on planar graphs MAXIMUM CONFLUENT FLOW is $\mathcal{NP}$-hard (see Theorem 2.13). In this chapter, we will examine confluent flows on more restrictive graph classes. We start with the easiest graph class for confluent flows, namely trees. Hereby, a directed graph is a tree if its underlying undirected graph is a tree. Afterwards, we will extend this result to graphs with *treewidth* bounded by a constant k and derive an approximation scheme for this class. We will also discover that it is easy to compute maximum confluent flows on *outerplanar graphs* and *planar graphs with all terminals on the boundary.* Several algorithms in this chapter are a result of joint work with Daniel Dressler and have already been published [53].

3.1 Confluent flows on trees

3.1.1 Maximum Confluent Flows on trees with a single sink

We have already seen that a maximum confluent S-T-flow on a tree with a single sink can be computed easily by a preflow-push algorithm (cf. Section 2.1.4). Still, there are some degrees of freedom in the choice of the next vertex to push. Since we will need a similar idea in Section 3.4, we properly specify Algorithm 3.1 for confluent S-T-flows on trees for later reference. For simplicity, we assume that the single sink has infinite capacity and that the tree is bi-directed.

Algorithm 3.1: CONFLUENTFLOWTREE

```
Input: bi-directed tree $G = (V, A)$, supplies $d : V \to \mathbb{N}_0$, capacities $u : A \to \mathbb{N}_0$, a
       sink $t \in V$ with $d(t) = 0$ and infinite capacity
Output: Maximum value of a confluent flow
1 Set $S := \{v \in V : d(v) > 0\}$;
2 Initialize $f \equiv 0$;
3 while $S \neq \emptyset$ do
4     choose $s \in S$ arbitrarily;
5     find the unique path $P$ from $s$ to $t$;
6     find bottleneck capacity of $P$, i.e. $b = \min_{e \in P} u(e) - f(e)$;
7     increase $f(e)$ along $P$ by $\min\{b, d(s)\}$;
8     $S := S \setminus \{s\}$;
9 return $\sum_{e \in \delta^-(t)} f(e)$
```

Lemma 3.1. *Algorithm 3.1 is correct and has polynomial runtime.*

Proof. Since G is a tree, the flow is confluent and there is a unique path from each source s to t. A (confluent) S-T-flow has to use this path if s should be used. The algorithm sends as much flow as possible from each source. A path may only be blocked by flow units from other sources considered earlier. However, their route choices were without alternative. Hence, the algorithm obviously determines the maximum flow value.

Furthermore, exploiting the tree structure of G, all steps of the algorithm can be performed in linear time. □

Algorithm 3.1 is obvious, one should keep in mind that the order of the sources has no influence on the flow value itself. But the flow distribution may change significantly when starting with another source.

Next, we show the existence of polynomial time algorithms for MAXIMUM CONFLUENT FLOW on trees if either the number of sources or sinks is bounded by a constant k. Recall that MAXIMUM CONFLUENT FLOW on trees with an unlimited number of sources *and* sinks is $\mathcal{NP}$-hard in general (see Theorem 2.13).

Theorem 3.2. MAXIMUM CONFLUENT FLOW *on trees can be solved in polynomial time if either the number of sinks or the number of sources is bounded by a constant.*

Proof. Let the number of sources or sinks be bounded by a constant k. In a confluent flow, the flow from each source travels along a path that eventually ends at one sink. As there are at most $|V|^k$ assignments of the sources to the sinks, and the paths taken by the flow units are uniquely determined, all possibilities can simply be enumerated and checked in polynomial time. □

As a simple extension, instances where the underlying undirected graph only differs from a tree by a constant number of edges can be reduced to a tree by "guessing" a suitable tree, i.e. with brute force.

3.1.2 Confluent transshipments on trees

MAXIMUM CONFLUENT FLOW on a tree with a single sink is an easy problem, because the optimal in-forest is obvious. But as shown in Section 2.4, MAXIMUM CONFLUENT FLOW with multiple sinks is weakly $\mathcal{NP}$-hard even on trees. Surprisingly, this does not hold for CONFLUENT TRANSSHIPMENT on trees, which can be solved in polynomial time, and we will use a finding of Chen et al. [30] to prove this.

Chen et al. studied confluent flows on trees with congestion and uniform capacities. They present a polynomial time algorithm which can be extended to an algorithm for CONFLUENT TRANSSHIPMENT with arbitrary capacities by slight modifications. The main idea is to resolve the confluent flow constraints at the leaves of the tree. For example, if there is a leaf that is a source node, its supply must leave along the only arc to the neighbor vertex.

Theorem 3.3. *There exists a polynomial time algorithm for* CONFLUENT TRANSSHIPMENT *on bidirected trees (with arbitrarily many sources and sinks).*

Proof. We assume that the tree is rooted at an arbitrary vertex. The algorithm proceeds to delete leaf after leaf, each time yielding a smaller equivalent instance of CONFLUENT TRANSSHIPMENT, until only a single vertex remains. The instance is a YES-instance if and only if this vertex has non-positive supply.

Suppose there is a leaf v which is not a sink. Let w denote the parent of v. All flow of v must be routed to w. If $u(v,w) < d(v)$, the flow cannot be routed and the instance is a No-instance. Otherwise, if w is not a sink we can increase $d(w)$ by $d(v)$ and delete v. If w is a sink, we decrease $c(w)$ by $d(v)$ and delete v. The instance is a No-instance if w does not provide enough capacity to do so.

In the remaining case, all leaves are sinks. We want to consider a vertex w in the second-lowest layer of the tree. All its children are leaves and therefore sinks. If w is a sink itself, we delete all of its children because by Definition 2.3 flow cannot leave a sink. Flow trying to reach them is absorbed in w. Otherwise, let v^* be a leaf below w that maximizes $\min\{u(w,v), c(v)\}$, i.e., v^* is a best sink among the children of w. Its effective capacity is $c^* := \min\{u(w,v), c(v^*)\} \geq 0$.

If $d(w) \leq c^*$, there is no need to route flow from w out of its subtree, so we can contract the arc (w,v). In terms of the algorithm, we set $c(w) := c^* - d(w) \geq 0$, i.e. w becomes a sink, and delete all the children of w. On the other hand, if $d(w) > c^*$, there is no possibility to route all the flow from w except upwards, so we leave $d(w)$ unchanged but again delete all children of w. □

The Proof of Theorem 3.3 suggests a polynomial time algorithm for Confluent Transshipment on trees. It is very important to choose a node from the second-lowest layer if all leaves are sinks. Otherwise, one can accidently push a sink into a path and by that block flow. See Figure 12 for an example.

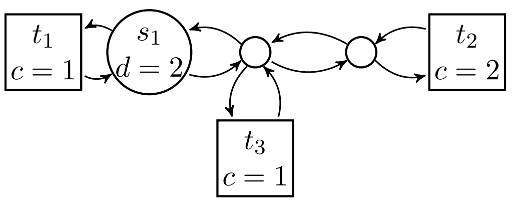

Figure 12: Instance of the Confluent Transshipment on a tree. The algorithm recursively resolves the leaves. Sink t_3 cannot be processed, because it would block the path from s_1 to t_2. Indeed, its parent node is also not on the second-lowest layer of the tree, whatever root was chosen. Thus, the algorithm continues with t_1 or t_2 in the next step.

3.2 Confluent flows on graphs with small treewidth

In the previous sections we have seen that confluent flows and trees go together. We will extend this result to graphs with *treewidth bounded by a constant k*. Treewidth can be considered to be a measure for how close a graph is to a tree. Trees are exactly those graphs with treewidth 1. Treewidth increases with the number of disjoint paths between nodes. The complete graph K_n has treewidth $n-1$. A precise definition will follow in the next Section 3.2.1.

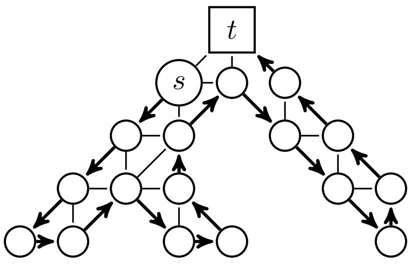

Figure 13: On a tree confluent flow always moves towards the sink. On a graph with bounded treewidth (in particular, the displayed graph has treewidth 2) flow may also move "away" from the sink taking a detour through the branches.

Treewidth is a prime example of parametrized complexity in graph theory. Normally, graphs with bounded treewidth are considered in the context of combinatorial problems like graph coloring or dominating set. In particular, each graph theoretical problem that can be expressed in terms of *monadic second order logic* can be parametrized by treewidth. This result was proven by Courcelle and Mosbah [38]. An alternative proof was presented by Arnborg, Lagergren, and Seese [8] at the same time.

Considering flow problems and bounded treewidth together is rather uncommon. Only a few examples can be found in the literature, e.g. an approach by Koch, Skutella, and Spenke to tackle k-splittable flows [92]. Approximation algorithms for multicommodity flows and treewidth were examined by Chekuri et al. [28]. Hagerup et al. study multi-terminal flows and external flow patterns [74]. Additionally, some problems like disjoint paths which are related to confluent flows were studied on those graphs [88, 134].

Recalling Corollary 2.14 Maximum Confluent Flow is at least weakly $\mathcal{NP}$-hard on graphs with treewidth 2. Therefore, we can only hope for a pseudo-polynomial time algorithm which leads to a polynomial time approximation scheme. Hence, we fall back to *dynamic programming*. Dynamic programming works well on graphs with bounded treewidth by exploiting the special structure of the underlying tree structure [18].

We will show that one can solve the Maximum Confluent Flow problem in a *bottom-up strategy* from the "leaves" to the "root". However, the algorithm is much more complicated than the corresponding algorithm for confluent flows on trees. Figure 13 illustrates one of the main difficulties.

3.2.1 Treewidth

For defining treewidth, we follow Bodlaender [19, 21]. For our purposes, the treewidth of a directed graph only depends on the underlying undirected graph with edge set $E := \{\{v, w\} : (v, w) \in A\}$, so the following definitions deal with undirected graphs. Johnson et al. [82] also give a generalization of treewidth to directed graphs, but it leads to the same results in the case that all reverse arcs are assumed to be in G as well.

Definition 3.4 (Tree decomposition). *Let $G = (V, E)$ be an undirected graph. A* tree decomposition *of G is a tree $(I, \mathcal{T})$ on some new node set I with edges $\mathcal{T}$, and a family of sets $B_i \subseteq V$ for $i \in I$. The following three properties must hold:*

1. $\bigcup_{i \in I} B_i = V$;
2. *for every edge $\{u, v\} \in E$, there is an $i \in I$ with $\{u, v\} \subseteq B_i$;*
3. *for all $i, j, k \in I$, if j lies on the unique path between i and k in $(I, \mathcal{T})$, then $B_i \cap B_k \subseteq B_j$.*

Each B_i is called a bag. *The* width *of a tree decomposition is* $\max_{i \in I} |B_i| - 1$. *The* treewidth *of G is the minimum k such that G has a tree decomposition of width k.*

The following additional requirements reduce the number of cases that have to be considered by the algorithm. Similar conditions are often used, e.g. [121]. In essence, they enforce that the bags change only slowly while traversing the tree.

Definition 3.5 (Nice tree decomposition). *Let $(I, \mathcal{T})$ with $\{B_i : i \in I\}$ be a tree decomposition of $G = (V, E)$ and let t be a vertex of V. Then this tree decomposition is called* nice *if $(I, \mathcal{T})$ is a binary tree for some root r, satisfying the following additional properties:*

4. *If a node $i \in I$ has two children $j, k \in I$, then $B_i = B_j = B_k$. In this case, i is called a* Join *node.*
5. *If a node $i \in I$ has one child j, then one of the following must hold:*
   a) *$|B_i| = |B_j| - 1$ and $B_i \subseteq B_j$. Then i is called a* Forget *node.*
   b) *$|B_i| = |B_j| + 1$ and $B_j \subseteq B_i$. Then i is called an* Introduce *node.*
6. *$|B_i| = 1$ holds for all leaves i of $(I, \mathcal{T})$. Such an i is called a* Leaf *node.*
7. *$B_r = \{t\}$ holds.*

Note that refinement *7* is a variation from the common definition in the literature, which further eases the construction of the algorithm. The root r can be chosen to contain an arbitrary vertex. We will use $B_r = \{t\}$, the (super-)sink. Consequently, we will follow Niedermeier [121] to prove the existence of a nice tree decomposition in Lemma 3.6.

Determining the treewidth of a graph is $\mathcal{NP}$-hard [7]. However, if the treewidth is bounded by a constant k, many related problems can be solved in polynomial or even linear time. In particular, one can check in linear time whether a graph has treewidth at most k. One can also construct a tree decomposition with width k in linear time, if it exists [20]. However, in all cases the hidden factors are large and grow quickly with k. See [21] for an overview of fast algorithms.

Lemma 3.6. *Let $G = (V, E)$ be a graph of treewidth k and $t \in V$ an arbitrary node. Then G has a nice tree decomposition $(I, \mathcal{T})$ of width k with root r such that $B_r = \{t\}$ and it can be constructed in linear time, assuming that k is constant. Furthermore, the number of nodes in I is polynomially bounded in the size of G.*

Proof. We assume a tree decomposition of polynomial size is given and make it "nice". Showing that the result remains a tree decomposition of G mostly revolves around condition 3 from above. Note that there is an equivalent formulation of condition 3, it requires that the nodes containing a given vertex v in their bags must form a connected component of $(I, \mathcal{T})$. This condition can be easily checked because we will only modify the tree locally.

First, we choose an arbitrary node r' with $t \in B_{r'}$ and orient the tree towards r'. If $|B_{r'}| > 1$, we introduce a new node r above r' that becomes the real root with $B_r = \{t\}$. If $|B_{r'}| = 1$, no such additional step is needed. Next we want to obtain a binary tree. We replace any node $i \in I$ with $p > 2$ children by a binary tree $(I', \mathcal{T}')$ with p leaves and all nodes $i' \in I'$ are associated with the vertex set B_i. The p children of i are then connected to the p children of the new binary tree $(I', \mathcal{T}')$. Note that the replacement binary tree can be chosen such that $|I'| \leq 2p - 1$. Thus, all such replacements can only double the size of the entire tree.

It is easy to see that leaves with empty bags may be deleted. Now consider a leaf i with $|B_i| > 1$. We simply add a new child node to it with a bag containing only one vertex out of B_i.

Notice that a single edge $\{i, k\}$ in the tree $(I, \mathcal{T})$ can be subdivided easily as long as the new node j between i and k has a bag satisfying $B_i \cap B_k \subseteq B_j \subseteq B_i \cup B_k$. So if i is a father of k that does not satisfy the bag size restrictions of a nice tree decomposition, this allows us to first introduce a node j with $B_j = B_i \cap B_k$ inbetween, and then "forget" all vertices in $B_i \setminus B_j$ one by one in further subdivisions of the edge $\{i, j\}$. Proceeding from j downwards, we can then "introduce" all vertices in $B_k \setminus B_j$ in subdivisions of $\{j, k\}$. For a JOIN node i it might actually be necessary to introduce one more subdivision to first obtain a child with the exact same bag B_i. The number of steps needed for each sequence of subdivisions is clearly bounded in $|V|$, giving a tree decomposition still with a polynomial number of nodes. Since we never use a bag larger than $\max\{|B_i|, |B_k|\}$, the treewidth does not increase. It only remains to contract edges where both ends have outdegree one and identical associated bags. $\square$

Additionally, we address the subgraph that is separated from t by the nodes in B_i as *bagend* G_i.

Definition 3.7 (Bagend). *Given a nice tree decomposition $(I, \mathcal{T})$ and $i \in I$, the* bagend *G_i is the subgraph of G induced by the union of B_i and all B_j for j in the subtree rooted at i.*

Let us establish some properties of nice tree decompositions that we will use in the proofs.

Lemma 3.8. *Let $(I, \mathcal{T})$ be a nice tree decomposition of $G = (V, E)$ with root r.*

1. *For a vertex $v \in V$, the set $\{i \in I : v \in B_i\}$ forms a connected component of $(I, \mathcal{T})$.*
2. *For a vertex $v \in V \setminus B_r$, there is a unique* FORGET *node $i \in I$ with child $j \in I$ such that $B_j = B_i \cup \{v\}$. There is no such* FORGET *node for $v \in B_r$.*

3. For a Join *node i with children j and k, it holds $B_i = V(G_j) \cap V(G_k)$.*

4. For an Introduce *node i with child j, let v be the unique vertex in $B_i \setminus B_j$. Then there are no edges between v and $V(G_j) \setminus B_j$.*

Proof. 1. This follows directly from Definition 3.4.3.

2. The unique Forget node for $v \in V \setminus B_r$ is the node $i \in I$ that contains v and is closest to the root r of $(I, \mathcal{T})$. If there was a Forget node for $v \in B_r$, that would contradict the connectedness of $\{i \in I : v \in B_i\}$.

3. Since $B_i = B_j = B_k$, we have $B_i \subseteq V(G_j) \cap V(G_k)$. Suppose there is a vertex $v \in V(G_j) \cap V(G_k) \setminus B_i$. Thus v is contained in some nodes j' and k' of the subtrees rooted at j and k, respectively, and so in any node on the path from j' to k'. In particular, $v \in B_i$, a contradiction.

4. Suppose there is an edge $\{v, w\}$ for some $w \in V(G_j) \setminus B_j$. This edge must be contained in some B_k, $k \in I$. Thus, v and w are in B_k, in particular $k \neq i$, $k \neq j$. If k is in the subtree of $(I, \mathcal{T})$ rooted j, then $v \in V(G_j)$, a contradiction. Otherwise the path from j to k passes through i. Then $w \in B_i$ must hold, again a contradiction.

□

3.2.2 The dynamic programming approach

We now explain the overall structure of our dynamic program for solving the Maximum Confluent Flow problem on graphs with treewidth bounded by a constant k. First, one pre-processes the instance of Maximum Confluent Flow to obtain one with finite supplies d and a single uncapacitated sink t. Then one determines a nice tree decomposition with t in the root bag. Such a tree decomposition has the useful property that each bag B_i of a node i separates the sink from the subgraph of G induced by all bags B_j of nodes j in the subtree of i, i.e. the bagend G_i. One can then compute suitable representations of all possible flows on each bagend, starting at the Leaf nodes and working towards the root. Having done so, it just remains to read off the maximum flow value at the root node.

The bags and bagends of a nice tree decomposition change towards the sink in an orderly fashion, so this really suggests a dynamic programming approach. However, one has to carefully choose representatives for the flows on each bagend G_i, while keeping enough information to develop the values for the next bag from these.

The main trick to make such a representation work is the following: The moment a vertex is first considered, one fixes the amount of flow that this vertex sends out. This is what we will call the *estimated outflow* $d^+(v)$. Of course, in a dynamic program we can try all possible integer values for each vertex up to a reasonable bound U and later on delete those combinations that turn out to be infeasible. For U we can take every upper bound on the flow value, the smaller the better. For example, one can choose the sum over all demands ($U = \sum_{v \in V} d(v)$) or the sum over the capacities of the incoming arcs of t ($U = \sum_{e=(v,t)} u(e)$).

Flow conservation requires that a vertex cannot send more flow than it receives (including its own supply), but when a vertex is first examined, it is not clear how much flow will eventually enter this vertex. Thus, flow conservation cannot be established immediately. Instead, we track how much more flow must reach a vertex in its *balance deficit*, which we denote by $d^-(v)$. The supply $d(v)$ does not come into play until the very last moment when a vertex leaves the currently considered bag.

The main advantage of fixing the estimated outflow is that we can locally manipulate the set of flow-carrying arcs. For example, if a vertex v uses no outgoing arc so far and we want to send the flow along (v, w), we can simply do so without triggering a chain reaction of increasing flow along the path that these flow units take. Indeed, such a path could lead into "uncharted territory" of the graph, which would make tracking the changes even harder. Instead, we simply update the balance deficit $d^-(w)$ of the receiving vertex. Its estimated outflow $d^+(w)$ remains unchanged, and therefore no other steps need to be taken.

We also need to keep track of some structural information for each B_i, the most obvious being the set of used arcs $F \subseteq A[B_i]$, which coupled with d^+ implies the actual flow on $A[B_i]$. On a very abstract level, one could misleadingly think of the following idea: a tree-like flow on a tree-like graph structure pointing towards the sink should only flow in the same direction as the underlying tree decomposition. However, this turns out to be wrong, and the optimum flow may send flow back and forth between B_i, the rest of the bagend $G_i \setminus B_i$ and the part with the sink $G \setminus G_i$ (cf. Figure 13). The implication is that we have to remember whether a vertex has an outgoing arc at all, because this might not be clear from F, if a flow-carrying outgoing arc points to an already "forgotten" vertex (an already processed vertex in $G_i \setminus B_i$). For this, each vertex $v \in B_i$ is assigned one of two states: $z(v) \in \{\textsc{free}, \textsc{nonfree}\}$. The state NONFREE denotes a vertex that already has an outgoing arc and satisfies the balance condition as if it had supply equal to its balance deficit $d^-(v)$. Vertices in the other state FREE resemble sinks, which is why they must have no outgoing arcs yet. These FREE vertices act as temporary storage. The estimated outflow $d^+(v)$ for a FREE vertex v means how much flow it *could* possibly send.

As can be expected for a problem of this complexity, the flow values that are possible at a single vertex are not independent of the flow values at a different vertex. Thus, the dynamic program will always consider tuples of these values as one entry of the dynamic program. All of this culminates in the following *table* for each bag B_i:

Definition 3.9 (Table). *Let $G = (V, A)$ be a graph with arc capacities $u : A \to \mathbb{N}_0$, supplies $d : V \to \mathbb{N}_0$ and an uncapacitated sink t. Assume that $\delta^+(t) = \emptyset$ and that $U \in \mathbb{N}_0$ is an upper bound on* MAXIMUM CONFLUENT FLOW *for this instance. Let $(I, \mathcal{T})$ and $(B_i)_{i \in I}$ be a nice tree decomposition of G.*

For $i \in I$, let $TABLE(i)$ be defined as the set containing exactly these elements (F, z, d^-, d^+) with $F \in \mathcal{F}(B_i)$, $z \in \{\textsc{free}, \textsc{nonfree}\}^{B_i}$, $d^- \in \{0, 1, \ldots, U\}^{B_i}$, $d^+ \in \{0, 1, \ldots, U\}^{B_i}$ for which there is an integral nearly confluent flow x in G_i with the *following properties:*

1. $a \in F \iff x(a) > 0 \qquad \forall a \in A[B_i]$
2. $d^+(v) = \text{inflow}_x(v) + d^-(v) \qquad \forall v \in B_i$
3. $\text{outflow}_x(v) = 0 \iff z(v) = \text{FREE} \qquad \forall v \in B_i$
4. $d^+(v) = \text{outflow}_x(v) \qquad \forall v \in B_i : z(v) = \text{NONFREE}$
5. $0 \leq \text{bal}(v) \leq d(v) \qquad \forall v \in G_i \setminus B_i$.

For brevity, $SET(i) := \mathcal{F}(B_i) \times \{\text{FREE}, \text{NONFREE}\}^{B_i} \times \{0, 1, \ldots, U\}^{B_i} \times \{0, 1, \ldots, U\}^{B_i}$ *denotes the domain of* $TABLE(i)$.

Please note that given a nearly confluent flow x on the bagend G_i and some table element (F, z, d^-, d^+), it is easy to check whether they match. However, x does not necessarily determine a unique entry in $TABLE(i)$. For a FREE vertex v, the flow only determines the difference $d^+(v) - d^-(v) = \text{inflow}_x(v)$ by condition 2, not the absolute values. For a NONFREE vertex, condition 4 eliminates any ambiguity.

A detailed view of this definition reveals that for a vertex v not all possible flow values in G_i are considered as $d^+(v)$ because the bound U on the overall flow value affects $d^+(v) \in \{0, 1, \ldots, U\}$ in any bag. Even if v could send or receive much more flow in G_i, this amount of flow could never reach the sink by the definition of U.

3.2.3 Computing the tables

For computing the entries of the $TABLE(i)$ out of the tables of the children of i, algorithms that address the four classes of nodes LEAF, INTRODUCE, JOIN, and FORGET are needed. One has to ensure that the flow values are calculated correctly and to proof that all entries can be derived in this way.

All of the following lemmata establish that certain entries are in $TABLE(i)$ based on the existence of a different entry in either $TABLE(i)$ or the table of the child or the children of i. Thus, we need to construct nearly confluent flows that match these new entries, starting with a flow for the old entry. Once a suitable flow has been chosen, it remains to check the conditions of Definition 3.9. Remember that flow just refers to a function without flow conservation. Flow conservation and demands are realized at the very end to turn a confluent flow into a confluent S-T-flow. Furthermore, we do not care about cycles, since the optimal value of a nearly confluent S-T-flow is equal to the value of a confluent S-T-flow (see Observation 2.9).

In these lemmata, we often modify vectors or functions and use a special notation for this. For example, if x is a flow, a an arc and $f \in \mathbb{R}$, we write $x \leftarrow (a : f)$ to denote a flow $\bar{x}$ almost like x except that $\bar{x}(a) = f$. Similarly, for a subset A' of arcs we use $x \leftarrow (A' : f)$ to set $\bar{x}(a) = f$ for all $a \in A'$. Note that this may extend the domain as well.

Leaf node. First, we present the easiest case of calculating $TABLE(i)$ of a LEAF node which only contains one vertex and cannot contain any arcs at all.

Lemma 3.10. *Let $i \in I$ be a* LEAF *node.*
Then $TABLE(i) = \{(\emptyset, (\text{FREE}), (f), (f)) : 0 \le f \le U\}$.

Proof. If i is a LEAF in $\mathcal{T}$, then $B_i = \{v\}$ by definition. Since G_i is just a single vertex, $A(G_i)$ is empty. We show the equality by showing both inclusions.

"$\supseteq$": Let $0 \le f \le U$. To show that $(\emptyset, (\text{FREE}), (f), (f))$ is in $TABLE(i)$ we have to consider some nearly confluent flow x and then check the conditions for $TABLE(i)$ given in Definition 3.9. Since $A(G_i) = \emptyset$, this flow x is a function on an empty set and thus there is no choice to be made. It is also trivially a nearly confluent flow. Note that $\emptyset$ is the set of flow-carrying arcs of $A(G_i)$, as required by condition 1. Also, $d^+(v) = f = d^-(v)$ and $\text{inflow}_x(v) = 0$, so condition 2 is satisfied. Since $\text{outflow}_x(v) = 0$, we set $z(v) = \text{FREE}$ in accordance with condition 3. Finally, there are no NONFREE vertices in this bag and no vertices in the empty set $G_i \setminus B_i$, either, so conditions 4 and 5 are trivially satisfied.

"$\subseteq$": Now let $(F, z, d^-, d^+) \in TABLE(i)$ and let x be a corresponding nearly confluent flow on G_i. (Again, x is defined on the empty set $A(G_i)$.) Since $F \subseteq A(G_i) = \emptyset$, we know $F = \emptyset$. Also, $\text{inflow}_x(v) = 0$ which, combined with condition 2, implies $d^+(v) = d^-(v) =: f \in \{0, 1, \dots, U\}$. Finally, condition 3 implies $z(v) = \text{FREE}$. Thus, every entry must have the form $(\emptyset, (\text{FREE}), (f), (f))$ as claimed and this concludes the proof.

□

Introduce node. The remaining types of nodes require a much more complex handling. Therefore, we sketch the basic ideas first. INTRODUCE nodes are algorithmically handled in two steps. First, the new vertex v is added to the bag as a FREE vertex without any flow entering or leaving it. Afterwards we can designate some of the arcs incident to v to carry flow to or from v, say along (w, w'), i.e. either w or w' is identical to v. This must *change* the state of w to NONFREE and it affects the balance of the receiving vertex w'. Of course, one needs to ensure that the arc capacity is not exceeded and that the receiving vertex actually requires that much additional incoming flow, i.e. $d^+(w) \le \min\{u(w, w'), d^-(w')\}$ must hold.

Lemma 3.11. *Consider an* INTRODUCE *node $i \in I$ with child j and v the unique vertex in $B_i \setminus B_j$. Let $(F, z, d^-, d^+) \in SET(j)$. Choose some $f \in \{0, \dots, U\}$. Then $(F, z, d^-, d^+) \in TABLE(j)$ if and only if*
$(F, z \leftarrow (v : \text{FREE}), d^- \leftarrow (v : f), d^+ \leftarrow (v : f)) \in TABLE(i)$.

Proof. $\Rightarrow$ Let x be a flow corresponding to $(F, z, d^-, d^+) \in TABLE(j)$. Let $\bar{x}$ be the extension of x to $A(G_i)$ with flow value 0. We claim that $\bar{x}$ matches the new entry of $TABLE(i)$ and show that the conditions of Definition 3.9 are met. Note that $\bar{x}$ is a nearly confluent flow. Also, F contains exactly those arcs of $A[B_i]$ that have non-zero flow (condition 1), because this was true for $A[B_j]$ and all arcs in $A[B_i] \setminus A[B_j]$ carry no flow. Condition 2 is still satisfied for $w \in B_j$ and, because $\text{inflow}_{\bar{x}}(v) = 0$ also for v by construction. Similarly, condition 3 is met by v, since v is set to be FREE. Conditions 4 and 5 are unaffected. Therefore, the claim holds.

$\Leftarrow$ Let x be a flow for $(F, z \leftarrow (v : \text{FREE}), d^- \leftarrow (v : f), d^+ \leftarrow (v : f)) \in \mathit{TABLE}(i)$ with $F \subseteq A[B_j]$ as this was implied by $(F, z, d^-, d^+) \in \mathit{SET}(j)$. Since $\{a \in A[B_i] : x(a) > 0\} = F \subseteq A[B_j]$, no flow can leave or enter v from B_j. By Lemma 3.8.4, there is no arc between v and $V(G_i) \setminus B_i$ either. Therefore, restricting x to $A(G_j)$ only removes arcs with flow 0 and thus, it results in a nearly-confluent flow again. Since the conditions for $\mathit{TABLE}(j)$ are a subset of those in $\mathit{TABLE}(i)$ and x is unchanged for the vertices in B_j, we can conclude $(F, z, d^-, d^+) \in \mathit{TABLE}(j)$.

□

Once the vertex is added, one can designate arcs that carry flow to v or away from v as follows. We will call sending flow across an arc like this a "push operation". Note that it applies to any kind of node, but we only use it for INTRODUCE nodes.

Lemma 3.12. *Let $i \in I$ and $(F, z, d^-, d^+) \in \mathit{SET}(i)$. Let $v \in B_i$ with $z(v) = \text{FREE}$ and consider some arc $(v, w) \in A(B_i) \setminus F$ such that $\bar{F} := F \cup \{(v, w)\}$ is in $\mathcal{F}(B_i)$. Suppose $0 < d^+(v) \leq \min\{u(v, w), d^-(w)\}$.*
Then $(F, z, d^-, d^+) \in \mathit{TABLE}(i)$ if and only if
$(\bar{F}, z \leftarrow (v : \text{NONFREE}), d^- \leftarrow (w : d^-(w) - d^+(v)), d^+) \in \mathit{TABLE}(i)$.

Proof. For brevity, let $\bar{z} := z \leftarrow (v : \text{NONFREE})$ and $\bar{d^-} := d^- \leftarrow (w : d^-(w) - d^+(v))$.

$\Rightarrow$ Let x be a flow on G_i that corresponds to $(F, z, d^-, d^+) \in \mathit{TABLE}(i)$. Let $\bar{x} := x \leftarrow ((v, w) : d^+(v))$. We claim that $\bar{x}$ is a nearly-confluent flow that shows $(\bar{F}, \bar{z}, \bar{d^-}, d^+) \in \mathit{TABLE}(i)$. Assuming $0 < d^+(v) \leq u(v, w)$, the function $\bar{x}$ is a flow and $\bar{F}$ are the arcs of $A[B_i]$ that carry flow. Also, since $z(v) = \text{FREE}$, there was no outgoing arc of v that carried flow before (v, w) was used, so $\bar{x}$ is a nearly confluent flow.
The conditions in Definition 3.9 on all vertices but v and w are unchanged, so it remains to check these two vertices. Condition 3 and $\text{outflow}_{\bar{x}}(v) = \bar{x}(v, w) = d^+(v) > 0$ forces $\bar{z}(v) = \text{NONFREE}$, while $\text{outflow}_{\bar{x}}(w)$ is unchanged. In fact, only $\text{outflow}_{\bar{x}}(v)$ and $\text{inflow}_{\bar{x}}(w)$ are changed. The change from $\text{outflow}_x(v) = 0$ to $\text{outflow}_{\bar{x}}(v) = d^+(v)$ matches condition 4. And $\text{inflow}_{\bar{x}}(w) + \bar{d^-}(w) = \text{inflow}_x(w) + d^+(v) + d^-(w) - d^+(v) = d^+(w)$ satisfies condition 2. Note that $\bar{d^-}(w) = d^-(w) - d^+(v)$ is non-negative by the assumption on $d^+(v)$.

$\Leftarrow$ Let $\bar{x}$ be the flow on G_i corresponding to $(\bar{F}, \bar{z}, \bar{d^-}, d^+) \in \mathit{TABLE}(i)$. We will set the flow on (v, w) to 0, i.e., let $x := \bar{x} \leftarrow ((v, w) : 0)$, and claim that x corresponds to (F, z, d^-, d^+). Clearly, x is a nearly confluent flow and F satisfies condition 1 of Definition 3.9. All vertices but v and w are again unaffected. For v, there is

no outgoing edge anymore and thus, $\text{outflow}_x(v) = 0$ matches $z(v) = \text{FREE}$. Condition 4 does not apply to v anymore and none of the values in condition 2 were changed for v, so it remains valid.
For w, most values are unaffected by the loss of the incoming edge, except $\text{inflow}_x(w)$. Also, the only entry of the table changed is $d^-(w)$. Since $\text{inflow}_x(w) + d^+(v) = \text{inflow}_{\bar{x}}(w) - \bar{x}(v,w) + d^+(v) = \text{inflow}_{\bar{x}}(w)$ by condition 4, we have $d^+(w) = \text{inflow}_{\bar{x}}(w) + \bar{d}^-(w) = (\text{inflow}_x(w) + d^+(v)) + (d^-(w) - d^+(v)) = \text{inflow}_x(w) + d^-(w)$, satisfying condition 2.

□

It remains to show that we can always construct all entries for an Introduce node with these two constructions.

Lemma 3.13. *Consider an* Introduce *node $i \in I$ with child j and v the unique vertex in $B_i \setminus B_j$. Let $(F, z, d^-, d^+) \in SET(i)$.*
Then $(F, z, d^-, d^+) \in TABLE(i)$ if and only if it can be produced by a sequence of push operations from some entry $(\bar{F}, \bar{z}, \bar{d}^-, \bar{d}^+) \in TABLE(i)$ with $\bar{F} \in \mathcal{F}(B_j)$, $\bar{z}(v) = \text{FREE}$ and $\bar{d}^-(v) = \bar{d}^+(v)$.

Proof. $\Leftarrow$ First note that if (F, z, d^-, d^+) is the result of such a sequence of push operations, then Lemma 3.12 applied repeatedly implies that it is an entry of $TABLE(i)$.

$\Rightarrow$ Let $(F, z, d^-, d^+) \in TABLE(i)$ and x be a corresponding flow. We use induction on $|F \setminus A[B_j]|$. If $|F \setminus A[B_j]| = 0$, then $F \in \mathcal{F}(B_j)$ and $z(v)$ must be free. Furthermore, by condition 2 in Definition 3.9, we have $d^+(v) = d^-(v) + \text{inflow}_x(v) = d^-(v)$, because no arcs incident to v carry flow. Thus, (F, z, d^-, d^+) is itself the desired entry.
Now assume $|F \setminus A[B_j]| > 0$. Pick an arc in $F \setminus A[B_j]$. This is either an arc (v, w) or (w, v), but we can treat it all the same as some (w, w'). Note that $z(w) = \text{NONFREE}$ must hold, which implies $d^+(w) = x(w, w')$ and $0 < x(w, w') \le u(w, w')$, because $(w, w') \in F$ and x is a flow. Let $\bar{x} := x \leftarrow ((w, w') : 0)$. Then $\bar{x}$ is again a nearly confluent flow. Let $\bar{F} := F \setminus \{(w, w')\}$, $\bar{z} = z \leftarrow (w : \text{FREE})$ and $\bar{d}^- \leftarrow (w' : d^-(w') + d^+(w))$. We claim that $\bar{x}$ justifies $(\bar{F}, \bar{z}, \bar{d}^-, d^+) \in TABLE(i)$. Clearly, $\bar{F}$ matches $\bar{x}$ and w must be FREE as its outflow is now 0. The only values affected by the change from x to $\bar{x}$ are $\text{outflow}_x(w)$ and $\text{inflow}_x(w')$. Since $\bar{z}(w) = \text{FREE}$, condition 4 does not matter for w anymore.
Note that $\text{inflow}_{\bar{x}}(w') = \text{inflow}_x(w') - x(w, w') = \text{inflow}_x(w') - d^+(w)$ and, thus, $d^+(w') = \text{inflow}_x(w') + d^-(w') = \text{inflow}_{\bar{x}}(w') + d^+(w) + d^-(w) = \text{inflow}_{\bar{x}}(w') + \bar{d}^-(w')$, satisfying condition 2 for w' again. Finally, $\bar{d}^-(w') \ge d^-(w') \ge 0$ and $\bar{d}^-(w') = d^-(w') + d^+(w) = d^+(w') - \text{inflow}_x(w') + d^+(w) \le d^+(w') - x(w, w') + d^+(w) = d^+(w) \le U$, thus the only changed value still obeys the boundaries. Therefore, we know $(\bar{F}, \bar{z}, \bar{d}^-, d^+) \in TABLE(i)$.
We want to apply Lemma 3.12 to the new entry $(\bar{F}, \bar{z}, \bar{d}^-, d^+)$, which will result in (F, z, d^-, d^+) again. However, we still need to check the conditions for this

push operation on (w, w') to finish the induction. We know $\bar{z}(w) = \text{FREE}$ and $(w, w') \in A(B_i) \setminus \bar{F}$ and that $\bar{F} \cup \{(w, w')\} = F$ is in $\mathcal{F}(B_i)$. Also, $d^+(w) = x(w, w') \in \{1, \dots, u(w, w')\}$ holds, as well as $d^+(w) \leq d^+(w) + d^-(w) = \bar{d}^-(w')$, so we are allowed to use the push operation.

□

This finally concludes the handling of Introduce nodes. Algorithm 3.2 presents a sample implementation.

```
Algorithm 3.2: INTRODUCE
Input: $i \in I$
Output: Array $TABLE(i)$
1  Set $j :=$ child of $i$;
2  Set $v$ to the unique vertex in $B_i \setminus B_j$;
3  foreach $(F, z, d^-, d^+) \in TABLE(j)$ do
4      foreach $f \in \{0, 1, \dots, U\}$ do
5          $\bar{z} := z \leftarrow (v : \text{FREE})$;
6          $\bar{d}^- := d^- \leftarrow (v : f)$;
7          $\bar{d}^+ := d^+ \leftarrow (v : f)$;
8          $TABLE(i) := TABLE(i) \cup (F, \bar{z}, \bar{d}^-, \bar{d}^+)$;
9  Initialize queue $Q = \emptyset$;
10 foreach $(F, z, d^-, d^+) \in TABLE(i)$ do
11     append $(F, z, d^-, d^+)$ to $Q$;
12 while $Q \neq \emptyset$ do
13     $(F, z, d^-, d^+)$ :=pop($Q$);
14     foreach $w \in B_i$ with $z(w) = \text{FREE}$ do
15         foreach $(w, w') \in A[B_i]$ do
16             $\bar{F} := F \cup \{(w, w')\}$;
17             $\bar{z} := z \leftarrow (w : \text{NONFREE})$;
18             $\bar{d}^- := d^- \leftarrow (w' : d^-(w') - d^+(w))$;
19             if $0 < d^+(w) \leq u(w, w') \wedge \bar{d}^-(w') \geq 0$ then
20                 if $(\bar{F}, \bar{z}, \bar{d}^-, d^+) \notin TABLE(i)$ then
21                     append $(\bar{F}, \bar{z}, \bar{d}^-, d^+)$ to $Q$;
22                     $TABLE(i) := TABLE(i) \cup (\bar{F}, \bar{z}, \bar{d}^-, d^+)$;
23 return $TABLE(i)$;
```

Join node. For a Join node i with children j and k, we "sum" selected entries of $TABLE(j)$ and $TABLE(k)$ in such a way, that one of the children carries no flow at all on the bag $B_i = B_j = B_k$. Note that the bagends G_j and G_k are disjoint elsewhere.

Thus, we do not have to worry about exceeding capacities there, or changing the already fixed value d^+.

Lemma 3.14. *Let $i \in I$ be a* Join *node with children $j, k \in I$. Let $(\bar{F}, \bar{z}, \bar{d}^-, \bar{d}^+) \in SET(i)$.*

Then $(\bar{F}, \bar{z}, \bar{d}^-, \bar{d}^+) \in TABLE(i)$ if and only if there are entries $(F, z, d^-, d^+) \in TABLE(j)$ and $(F', z', d'^-, d'^+) \in TABLE(k)$ satisfying

a) $F = \bar{F}$ and $F' = \emptyset$,

b) $z(v) = \text{FREE}$ or $z'(v) = \text{FREE} \qquad \forall v \in B_i$,

c) $\bar{z}(v) = \text{FREE} \iff z(v) = z'(v) = \text{FREE} \qquad \forall v \in B_i$,

d) $\bar{d}^+ = d^+ = d'^+$

e) $\bar{d}^- = d^- + d'^- - d^+$.

Proof. $\Leftarrow$ Let x and x' be the flows on G_j and G_k, respectively, corresponding to the entries in $TABLE(j)$ and $TABLE(k)$. Note that G_j and G_k are subgraphs of G_i and extend those flows to G_i with 0. Let $\bar{x} := x + x'$. We claim that $\bar{x}$ is the desired flow. First of all note that by Lemma 3.8 part 3 the bagends G_j and G_k share only the vertices of B_i and in particular they can only share the edges $A[B_i]$. Since $F' = \emptyset$, all of x' is 0 on $A[B_i]$. Thus, each edge of G_i can only carry positive flow in one of x and x', so $\bar{x}$ obeys the capacity constraints. Similarly, $\bar{x}$ is nearly confluent. If it was not, some vertex in B_i would have two outgoing arcs carrying flow. This cannot happen with just the edges from x or x', so the vertex was NONFREE in both flows. This contradicts the condition of the lemma that $z(v) = \text{FREE}$ or $z'(v) = \text{FREE}$ for all vertices.
It remains to show that $\bar{x}$ matches $(F, \bar{z}, \bar{d}^-, d^+)$, i.e. the conditions of Definition 3.9. (Note that $F = \bar{F}$ and $d^+ = \bar{d}^+$, so this is the desired entry.) Only x contributes flow on $A[B_i]$ to $\bar{x}$ and F is also derived entirely from x, so condition 1 is satisfied. Since $\text{outflow}_{\bar{x}}(v) = \text{outflow}_x(v) + \text{outflow}_{x'}(v)$, this is 0 iff $z(v) = z'(v) = \text{FREE}$ and in exactly this case $\bar{z}(v) = \text{FREE}$ as well, as demanded by condition 3. Also, $\text{outflow}_x(v) > 0$ implies $z(v) = \text{NONFREE}$, thus $z'(v) = \text{FREE}$ leading to $\text{outflow}_{x'}(v) = 0$ and vice versa. So for a vertex with $\bar{z}(v) = \text{NONFREE}$, we obtain $d^+(v) = \text{outflow}_x(v) + \text{outflow}_{x'}(v) = \text{outflow}_{\bar{x}}(v)$ as required in condition 4. Now consider condition 2 for some $v \in B_i$: $d^+(v)$ is equal for both table entries and also for the new entry. Then $\text{inflow}_{\bar{x}}(v) = \text{inflow}_x(v) + \text{inflow}_{x'}(v) = d^+(v) - d^-(v) + d'^+(v) - d^+(v) = d^+(v) - \bar{d}^-(v)$. So this condition is satisfied as well. Finally, the balance conditions for the vertices in $G_i \setminus B_i$ are carried over from the two separate flows.

$\Rightarrow$ Let $\bar{x}$ be the flow on G_i corresponding to $(\bar{F}, \bar{z}, \bar{d}^-, \bar{d}^+) \in TABLE(i)$. We need to construct appropriate flows x and x' on G_j and G_k. Choose $x := \bar{x}|_{A(G_j)}$ as the restriction of $\bar{x}$ to the arc set of G_j. Similarly, choose $x' := \bar{x}|_{A(G_k)} \leftarrow (A[B_i] : 0)$ as the restriction of $\bar{x}$ to G_k, but set to 0 on all arcs in B_i. Note that $\bar{x} = x + x'$

if x and x' are extended with 0 to the entire arc set of G_i, because Lemma 3.8.3 shows that G_j and G_k have exactly the vertices of B_i and in particular the arcs in $A[B_i]$ in common.

Clearly, x and x' are nearly confluent flows satisfying the capacity conditions, and so they correspond to some entries in $(F, z, d^-, d^+) \in TABLE(j)$ and $(F', z', d'^-, d'^+) \in TABLE(k)$. Actually, there are many choices for these entries, as a flow does not determine d^+ and d^- for a FREE vertex, but only their difference. We can assume that $d^+(v) = \bar{d}^+(v)$ for all $v \in B_i$ with $z(v) = \text{FREE}$, as long as we ensure that $d^+(v)$ and $d^-(v)$ stay within $0, 1, \ldots, U$. If we increase $d^+(v)$, there is no problem because $d^-(v) \leq d^+(v) = \bar{d}^+(v) \leq U$. If we had to decrease $d^+(v)$, note that then for the new values we have $d^-(v) = d^+(v) - \text{inflow}_x(v) \geq \bar{d}^+(v) - \text{inflow}_{\bar{x}}(v) = \bar{d}^-(v) \geq 0$, so this is also unproblematic. Similarly, we can assume $d'^+(v) = \bar{d}^+(v)$ for all $v \in B_i$ with $z'(v) = \text{FREE}$.

We can now show that entries chosen like this satisfy the conditions claimed in the lemma.

a) x equals $\bar{x}$ on $A[B_i]$ and so $F = \bar{F}$ holds. The flow x' is 0 on $A[B_i]$, implying $F' = \emptyset$.

b) Suppose for some $v \in B_i$, $z(v) = z'(v) = \text{NONFREE}$. Since $z(v) = \text{NONFREE}$, there is some arc (v, w) that carries flow in G_j. The same arc (v, w) carries flow in $\bar{x}$. Similarly, $z'(v) = \text{NONFREE}$ implies the existence of an arc (v, w') that carries flow in G_k and G_i. However, this cannot be the same arc as (v, w) because $A(G_j) \cap A(G_k) = A[B_i]$, but x' is 0 on $A[B_i]$. But then flow leaves v via two arcs in $\bar{x}$, a contradiction.

c) Note that $\bar{z}(v) = \text{FREE}$ iff there is no flow-carrying arc leaving v in $\bar{x}$. Since $\bar{x} = x + x'$, this is equivalent to neither x nor x' having flow-carrying arcs leaving v. By the definition of $TABLE$, this holds iff $z(v) = z'(v) = \text{FREE}$.

d) For a vertex v with $z(v) = \text{FREE}$, we have $d^+(v) = \bar{d}^+(v)$ by construction, and similarly $z' = \text{FREE}$ implies that we chose $d'^+(v) = \bar{d}^+(v)$. Now suppose $z(v) = \text{NONFREE}$, which implies $z'(v) = \text{FREE}$ and $\bar{z}(v) = \text{NONFREE}$ as we just proved. Then x must have a flow carrying arc (v, w), while $x'(v, w) = 0$. Thus, $\bar{d}^+(v) = \bar{x}(v, w) = x(v, w) + x'(v, w) = x(v, w) = d^+(v)$. Analogously, $z'(v) = \text{NONFREE}$ implies $z(v) = \text{FREE}$ and $\bar{z}(v) = \text{NONFREE}$, and again $\bar{d}^+(v) = \bar{x}(v, w) = x'(v, w) = d'^+(v)$.

e) By definition of $TABLE(i)$, $\bar{d}^-(v) = \bar{d}^+(v) - \text{inflow}_{\bar{x}}(v)$, which decomposes into $\bar{d}^-(v) = d^+(v) - \text{inflow}_x(v) - \text{inflow}_{x'}(v) = d^-(v) - (d^+(v) - d^-(v))$ as claimed, using the definition of $TABLE(j)$ and $TABLE(k)$ and the just proven property $\bar{d}^+ = d^+ = d'^+$.

□

Again, we present a sample implementation in Algorithm 3.3.

Algorithm 3.3: JOIN

```
Input: $i \in I$
Output: Array $TABLE(i)$
Set $j :=$ first child of $i$;
Set $k :=$ second child of $i$;
foreach $(F, z, d^-, d^+) \in TABLE(j)$ do
    foreach $(\emptyset, z', d'^-, d^+) \in TABLE(k)$ do
        Set $ok :=$ true;
        foreach $v \in B_i$ do
            if $z(v) = \text{FREE} \wedge z'(v) = \text{FREE}$ then
                $\bar{z}(v) := \text{FREE}$;
            else if $z(v) = \text{FREE} \vee z'(v) = \text{FREE}$ then
                $\bar{z}(v) := \text{NONFREE}$;
            else
                $ok :=$ false;
            $\bar{d}^-(v) := d^-(v) + d'^-(v) - d^+(v)$;
            if $\bar{d}^-(v) < 0$ then
                $ok :=$ false;
        if $ok = true$ then
            $TABLE(i) := TABLE(i) \cup \{(F, \bar{z}, \bar{d}^-, d^+)\}$;
return $TABLE(i)$;
```

Forget node. It remains to describe the process for Forget nodes. In this case, the vertex v that leaves the bag suddenly needs to satisfy any remaining imbalance $d^-(v)$ with its own supply $d(v)$, if it uses an outgoing arc at all. Otherwise, it should not try to send any flow.

Fortunately, Forget nodes are easy to handle. As long as the vertex that left the bag satisfies the flow conservation constraint, either by using its own supply $d(v)$ or by not being part of the flow at all, everything works out fine.

Lemma 3.15. *Let $i \in I$ be a* Forget *node and $j \in I$ its unique child. Let $B_j \backslash B_i = \{v\}$. Let $(F, z, d^-, d^+) \in SET(i)$.*

Then $(F, z, d^-, d^+) \in TABLE(i)$ if and only if there is an entry $(\bar{F}, \bar{z}, \bar{d}^-, \bar{d}^+) \in TABLE(j)$ with $F = \bar{F} \cap A[B_i]$, $z = \bar{z}|_{B_i}$, $d^- = \bar{d}|^-_{B_i}$ and $d^+ = \bar{d}|^+_{B_i}$, satisfying one of the following conditions:

(i) $\bar{z}(v) = \text{FREE}$ *and* $\bar{d}^+(v) = 0$ *or*

(ii) $\bar{z}(v) = \text{NONFREE}$ *and* $\bar{d}^-(v) \le d(v)$.

Proof. $\Leftarrow$ Let x be a flow in G_j corresponding to $(\bar{F}, \bar{z}, \bar{d}^-, \bar{d}^+) \in TABLE(j)$ and satisfying the assumptions of the lemma. Note that $G_i = G_j$, even though v is not in B_i. Then x can also be considered as a nearly confluent flow on G_i. It remains to show that it matches the entry of $TABLE(i)$ as claimed. However, x is not changed and the conditions of $TABLE(i)$ are a subset of the conditions of $TABLE(j)$, the only exception being the flow conservation constraint for v. First suppose $\bar{z}(v) =$ FREE, i.e., no flow leaves v, and $\bar{d}^+(v) = 0$. Then $\text{inflow}_x(v) + \bar{d}^-(v) = \bar{d}^+(v) = 0$, so there is no flow entering v either. Thus, v satisfies the flow conservation constraint with $\text{bal}_x(v) = 0$. Otherwise, the lemma assumed $z(v) =$ NONFREE and $\bar{d}^-(v) \leq d(v)$. This implies $\text{outflow}_x(v) = \bar{d}^+(v) = \text{inflow}_x(v) + \bar{d}^-(v) \leq \text{inflow}_x(v) + d(v)$. So $\text{outflow}_x(v) - \text{inflow}_x(v) \leq d(v)$. Also, $\text{outflow}_x(v) = \text{inflow}_x(v) + \bar{d}^-(v) \geq \text{inflow}_x(v)$ shows the other side of the flow conservation constraint.

$\Rightarrow$ Let x be a flow on G_i corresponding to $(F, z, d^-, d^+) \in TABLE(i)$. Using $G_i = G_j$ again, we consider x on G_j and claim that it is the desired flow. Choose $\bar{F} \subseteq A[B_j]$ according to x. All entries of $\bar{z}, \bar{d}^-, \bar{d}^+$ can be chosen for $w \in B_i$ exactly as in z, d^-, d^+ as they only depend on x. Note that this already satisfies all the properties claimed in the lemma except those for v, where we have not fixed any values yet.

If $\text{inflow}_x(v) = \text{outflow}_x(v) = 0$, let $\bar{z}(v) :=$ FREE and $\bar{d}^+(v) := \bar{d}^-(v) := 0$. This also satisfies the conditions of $TABLE(j)$ for v. Otherwise, let $\bar{z}(v) :=$ NONFREE and $\bar{d}^+(v) := \text{outflow}_x(v)$ and $\bar{d}^-(v) := \text{bal}_x(v) = \text{outflow}_x(v) - \text{inflow}_x(v)$. Since x satisfied $0 \leq \text{bal}_x(v) \leq d(v)$, this satisfies condition 2 and 4 of Definition 3.9. Furthermore, this yields an entry as claimed in the lemma.

$\square$

We also present an implementation in Algorithm 3.4.

Algorithm 3.4: FORGET

```
Input: $i \in I$
Output: Array $TABLE(i)$
1 Set $j :=$ child of $i$;
2 Set $v$ to the unique vertex in $B_j \setminus B_i$;
3 foreach $(F, z, d^-, d^+) \in TABLE(j)$ do
4     if $z(v) = \text{FREE} \wedge d^+(v) = 0$ then
5         $TABLE(i) := TABLE(i) \cup \{(F \cap A[B_i]), z|_{B_i}, d^-_{B_i}, d^+_{B_i}\}$;
6     else if $z(v) = \text{NONFREE} \wedge d^-(v) \leq d(v)$ then
7         $TABLE(i) := TABLE(i) \cup \{(F \cap A[B_i]), z|_{B_i}, d^-_{B_i}, d^+_{B_i}\}$;
8 return $TABLE(i)$;
```

3.2.4 Putting it all together

Well-equipped with algorithms for the four node classes, one can then formulate a dynamic program that computes all tables, represented as arrays containing $|SET(i)| \in \mathcal{O}(2^{(k+1)^2} 2^{k+1} U^{2(k+1)}) = \mathcal{O}(U^{2k+2})$ bits. Thus, we obtain the following theorem.

Theorem 3.16. *There is an algorithm that given a nice tree decomposition $(I, \mathcal{T})$ computes all tables correctly and runs in time polynomial in $|I|$ and U, assuming that the width k of $(I, \mathcal{T})$ is constant.*

Proof. It is clear that computing the tables bottom-up will result in $|I|$ calls to the subroutines handling the different cases of nodes in the tree decomposition. It just remains to show that the running time is as claimed. We will represent each table as a binary array of size $|SET(i)|$, which is polynomial in $|U|$ if k is a constant.

1. For a Leaf node, Lemma 3.10 gives an explicit statement of $TABLE(i)$. This requires $\mathcal{O}(|U|)$ insertions.

2. For an Introduce node, an algorithm would first have to insert the new vertex as a free vertex for any entry of the existing table (Lemma 3.11). This requires $\mathcal{O}(|SET(j)||U|)$ insertions. Starting from this table, a breadth-first search can be employed to produce all combinations of pushing flow across arcs as in Lemma 3.12. Every entry has to be considered only once in the BFS, and can produce at most $|A[B_i]| \leq k(k+1)$ new entries. Lemma 3.13 shows that this creates all entries.

3. For a Join node, the algorithm can simply loop over all combinations of pairs from the tables of the children that match the conditions of Lemma 3.14. Only if they are all satisfied, the entry is inserted into $TABLE(i)$. This requires $\mathcal{O}(|SET(j)||SET(k)|)$ checks. Additionally, Lemma 3.14 also states that such two entries always exist to create an entry of $TABLE(i)$.

4. For a Forget node, the algorithm applies the rules from Lemma 3.15 to produce only valid entries. The same lemma also shows that these are all entries that need to be considered. This is a simple projection for $\mathcal{O}(|SET(j)|)$ entries.

Thus, the algorithm computes each table in time polynomial in $|U|$, proving the overall running time to be polynomial in $|I|$ and $|U|$. □

Algorithm 3.5 is a sample implementation for the process described in Theorem 3.16.

We can now apply the previous theorem to solve the Maximum Confluent Flow problem as follows.

Theorem 3.17. Maximum Confluent Flow *on graphs with an arbitrary number of sinks and with treewidth bounded by some constant k can be solved in pseudo-polynomial time.*

Algorithm 3.5: MaximumConfluentFlowBoundedTreewidth

```
Input: $G = (V, A)$, supplies $d : V \to \mathbb{N}_0$, capacities $u : A \to \mathbb{N}_0$, a sink $t \in V$ with
       $\delta^+(t) = \emptyset$, a bound on the maximum confluent flow value $U$, a nice tree
       decomposition $(I, \mathcal{T})$ with bags $B_i$ of $G$, root $r$ and $B_r = \{t\}$
Output: $TABLE(i)$ for each $i \in I$ according to Definition 3.9
1  Initialize empty arrays $TABLE(i)$ for all $i \in I$;
2  Set $J := I$;
3  while $J \neq \emptyset$ do
4      Choose $i \in J$ such that all children of $i$ are not in $J$;
5      if $i$ has 0 children then
6          $TABLE(i) := \{(\emptyset, (\text{FREE}), (f), (f)) : 0 \leq f \leq U\}$;
7      else if $i$ has 1 child $j$ and $|B_i| = |B_j| + 1$ then
8          $TABLE(i) :=$ INTRODUCE(i);
9      else if $i$ has 1 child $j$ and $|B_i| = |B_j| - 1$ then
10         $TABLE(i) :=$ FORGET(i);
11     else if $i$ has 2 children then
12         $TABLE(i) :=$ JOIN(i);
13     $J := J \setminus \{i\}$;
14 return $TABLE(i)$ for each $i \in I$
```

Proof. Consider an instance $G = (V, A)$, u, d, S, T and c. It is assumed that the treewidth of G is at most k. W.l.o.g. we can assume that no vertex has larger supply than it could send to a sink by itself. This can be assured in polynomial time by solving the BottleneckShortestPath problem, e.g., with a modified Dijkstra's algorithm (cf. [83]). In particular, these values are finite.

First we preprocess the instance to obtain a graph G' with a single supersink t^*: Remove all arcs leaving the sinks. Then add a supersink t^* and connect each old sink t to it by an arc (t, t^*) with capacity $\min\{c(t), \sum_{(v,t)\in A} u(v, t)\}$. The supersink has infinite sink capacity and no supply, while the old sinks T now become normal vertices, i.e. $T' = \{t^*\}$. Notice that this preprocessing does not change the solution of Maximum Confluent Flow since sinks are not allowed to send out flow to any other vertex. Also, the treewidth of this new instance is at most $k + 1$ because a tree decomposition of G can be turned into a tree decomposition of G' by simply adding t^* to every bag.

Next we compute a nice tree decomposition of G' with width at most $k + 1$ and with polynomially many nodes, as described in Lemma 3.6. Finally, we run the dynamic program from Theorem 3.16 on the modified instance with $U := \sum_{v\in V} d(v)$ to compute $TABLE(r)$ for $B_r = \{t^*\}$. Note that the overall running time is pseudo-polynomial in the size of G and its demands and capacities.

Now, we claim that the solution is the maximum value $f \in \{0, 1, \ldots, U\}$ such that $(\emptyset, (\text{FREE}), (0), (f)) \in TABLE(r)$. This value can be extracted from the table easily. To see that it is indeed the solution, first recall that the maximum value of a nearly confluent S-T'-flow is the same as the maximum value of a confluent S-T'-flow (Observation 2.9).

Note that $G_r = G$ and that every nearly confluent S-T'-flow x induces an entry in $TABLE(r)$ of the form $(\emptyset, (\textsc{free}), (0), (\mathrm{bal}_x(t^*)))$. On the other hand, all table entries of this kind correspond to a nearly confluent flow x which satisfies the flow conservation constraint in every vertex except t^*. Since $d^-(t^*) = 0$, we have $d^+(t^*) = \mathrm{inflow}_x(t^*)$ and $\mathrm{outflow}_x(t^*) = 0$ because $z(t^*) = \textsc{free}$, so the value of the flow must be equal to $d^+(t^*)$. □

3.2.5 Constructing an FPTAS

The previous results can be extended to an FPTAS for Maximum Confluent Flow.

Theorem 3.18. Maximum Confluent Flow *on graphs with $|V|$ nodes and treewidth bounded by some constant k can be approximated within a factor of $(1-\varepsilon)$ for any $\varepsilon > 0$ in time polynomial in $|V|$ and $1/\varepsilon$.*

Proof. Again the input is $G = (V, A)$, u, d, T and c and we assume that the supplies d are all individually routable. We first have to bound the value f^* of an optimal solution. With $d_{\max} := \max_{v \in V} d(v)$, we obtain $d_{\max} \leq f^* \leq \sum_{v\in V} d(v) \leq |V| d_{\max} =: U$.

We will use a simple rounding scheme to obtain the approximation. Let $\varepsilon' := \varepsilon/|V|^2$ and round down all numerical values to multiples of $\varepsilon' U$. Call these new values $\tilde{u}$, $\tilde{d}$ and $\tilde{c}$. For example, $\tilde{u}(a) := \left\lfloor \frac{u(a)}{\varepsilon' U} \right\rfloor \varepsilon' U$. If we further scale the instance by $(\varepsilon' U)^{-1}$, we obtain integral values between 0 and $\lfloor 1/\varepsilon' \rfloor$. We run the algorithm to solve this scaled down integral instance. Theorem 3.17 shows that the running time is polynomial in the size of the graph and $\tilde{U} := (\varepsilon' U)^{-1} \sum_{v \in V} \tilde{d}(v) \in \mathcal{O}(|V|/\varepsilon') = \mathcal{O}(|V|^3/\varepsilon)$. Thus, this is a polynomial time algorithm overall.

Let f denote the optimum value of the rounded instance scaled back to the original range by a factor of $\varepsilon' U$. (The optimum value scales with the parameters.) We need to show $f \geq (1-\varepsilon) f^*$ to conclude this proof.

Consider a confluent S-T-flow x with value f^* for the original instance. Standard flow theory tells us that we can decompose it into at most $|F_x| \leq |V|$ paths $P \in \mathcal{P}$ carrying flow from the sources to the sinks. Cycles do not occur in F_x by definition. We denote the flow on each path with $x(P)$, and $\sum_{P \in \mathcal{P}} x(P) = f^*$ must hold. Rounding the flow on each path to $\tilde{x}(P) := \left\lfloor \frac{x(P)}{\varepsilon' U} \right\rfloor \varepsilon' U$ yields a path decomposition of a new flow $\tilde{x}$ where all flow values are multiples of $\varepsilon' U$. Rounding down individual paths results in a larger loss due to rounding than for d, u or c, so $\tilde{x}$ obeys all rounded capacities. Thus, $\tilde{x}$ is a confluent S-T-flow that is feasible in the rounded instance and has value $\sum_{P \in \mathcal{P}} \tilde{x}(P) \geq \varepsilon' U \sum_{P\in\mathcal{P}} (\frac{x(P)}{\varepsilon' U} - 1) = f^* - |V| \varepsilon' U$. Since f is the optimum value of a feasible flow in the rounded instance, it follows that $f \geq \mathrm{val}(\tilde{x}) \geq f^* - |V|\varepsilon' U$.

Recalling the bounds on f^*, we obtain $f \geq f^* - |V|\varepsilon' U = f^* - |V|^2 \varepsilon' d_{\max} = f^* - \varepsilon d_{\max} \geq f^* - \varepsilon f^* = (1-\varepsilon) f^*$ as claimed. □

As usual in dynamic programming, backtracking provides the solution, i.e. the actual S-T-flow, for both algorithms. Note that it is also possible to modify the exact dynamic program in order to handle, e.g., lower capacities on arcs or all-or-nothing flows. However, since the accuracy of the rounded instance is limited, these conditions only carry over in an $\mathcal{O}(\varepsilon)$-precise way to the approximation algorithm, i.e. lower bounds may be violated by at most $\varepsilon' U$ since all numerical values are rounded down.

3.3 Confluent flows on outerplanar graphs

One observation encourages to reconsider the gap between easy and hard confluent flow problems despite the hardness results on trees: On cycles with a single sink Maximum Confluent Flow can be solved efficiently. There are only $|V|$ possibilities to split the flow – the flow on one side of this split point reaches the sink clockwise, the flow on the other side moves counterclockwise.

We will now extend this approach to *outerplanar* graphs, which look “somewhat like cycles” but are complex enough to foil a brute force approach. The ideas presented in this section fall into two groups. First, we explore the hierarchy of a certain class of self-contained induced subgraphs to enable dynamic programming. Second, we analyze and exploit the properties of the dynamic programming solutions to achieve polynomial running times by expressing the data in a compressed way.

3.3.1 Outerplanar graphs

Definition 3.19 (Outerplanar graph). *An undirected graph $G = (V, E)$ is called* outerplanar*, if it has an embedding in the plane such that all vertices lie on the outer face. We will call a digraph outerplanar if the underlying undirected graph is outerplanar.*

Outerplanar graphs are exactly those graphs that do not have K_4 or $K_{2,3}$ as minor [27] and they can be recognized in linear time [114]. The usual drawing of an outerplanar graph places all vertices on a circle and all edges are embedded within that circle (Figure 14). A fascinating result [72] shows that given any outerplanar graph $G = (V, E)$ and any set of points P in the plane in general position (i.e., no 3 points lie on the same straight line) with $|P| = |V|$, one can find an embedding of G such that the vertices are mapped to P and all edges are represented by straight lines. Actually, with this result outerplanar graphs can be easily seen to be the largest graph class that admits such an embedding for every given point set. There is an $\mathcal{O}(|V| \log^3 |V|)$ algorithm to find such an embedding, and a simpler $\mathcal{O}(|V|^2)$ algorithm exists as well [22]. We will use this results and always embed the outerplanar graph on a (regular) n-gon.

3.3.2 Solving confluent flows on outerplanar graphs

In this section we will explore the intrinsic structure of outerplanar graphs that limits the number of choices for the single sink case of Maximum Confluent Flow that need to be considered by a dynamic program.

We start with a high-level description of the algorithm. As mentioned above we can assume that we are given a straight-line embedding of a bidirected outerplanar graph $G = (V, A)$ on the regular n-gon, and that the vertices are labelled clockwise v_0 to v_{n-1}. We may also assume that the entire outer cycle is part of the graph in both directions. Adding these arcs with capacity 0 cannot lead to any crossings by simple geometric

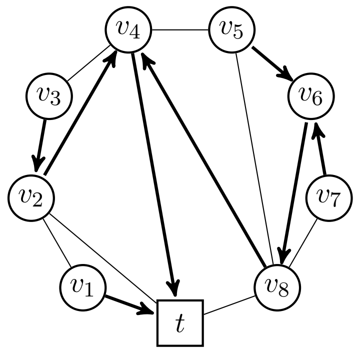

Figure 14: A maximal outerplanar graph and a possible spanning in-tree. An outerplanar graph consists of up to $2|V|-3$ edges (or $4|V|-6$ arcs respectively) and therefore contains an exponential number of spanning in-trees. The chords admit *zigzag paths* for flow units.

arguments. An arc (v_i, v_j) with $|i-j| > 1$ is called *chord*. Since we only consider the single sink case, let the (unlimited) sink be v_0, located at the bottom of the cycle. For the sake of brevity, we think of vertices with low indices as being left of vertices with higher indices instead of saying "in counterclockwise direction".

Our dynamic program works on "slices" cut out of the cycle. More precisely, for $i < j$, we consider the graph induced by $\{v_i, v_{i+1}, \dots, v_j\}$, which we denote by $G_{i,j}$. This is again an outerplanar graph (with the same embedding as in G). We will only consider subgraphs $G_{i,j}$ that are *self-contained*, i.e. no vertex v_k with $i < k < j$ is adjacent to a vertex outside of $G_{i,j}$.

Definition 3.20. *$G_{i,j}$ denotes the* self-contained *induced subgraph $G[\{v_i, v_{i+1}, \dots, v_j\}]$.*

We want to capture all the relevant information about the flow inside of $G_{i,j}$ in the dynamic program. Since $G_{i,j}$ is self-contained, we only need to know the balance at v_i and v_j. Once this information is available, we will join consecutive $G_{i,j}$ and $G_{j,k}$ to obtain the values for $G_{i,k}$.

Recalling treewidth and tree decompositions (cf. Definition 3.4), the subgraphs $G_{i,j}$ also have another interpretation. Outerplanar graphs are 2-trees and therefore they have treewidth at most 2 [24]. Let (v_i, v_j) be a chord in our embedding. Since every arc has to be contained in a bag by definition of treewidth, $\{v_i, v_j\}$ is a bag of every nice tree-decomposition of width 2 of the outerplanar graph. Remember, that bags only change slowly due to INTRODUCE and FORGET nodes. Therefore, the subgraph $G_{i,j}$ is equal to the bagend induced by $\{v_i, v_j\}$. However, we want to develop a polynomial time algorithm. Hence, the pseudo-polynomial dynamic program in Section 3.2.2 is not powerful enough. Furthermore, we have seen a graph with treewidth 2 in Figure 3, that was used in the $\mathcal{NP}$-hardness proof. That is, there is something special about outerplanar graphs that we have to exploit. This will be uncovered in Lemma 3.23. Hence, a tree decomposition and the results of Section 3.2 are of minor use here.

To determine the balance $\mathrm{bal}_x(v_i)$ and $\mathrm{bal}_x(v_j)$ for the dynamic program, consider an optimum confluent S-T-flow x on G. The flow-carrying arcs induce an in-tree pointing towards the sink which we can greedily extend to a spanning in-tree. The path from any vertex in $G_{i,j}$ to v_0 must either travel through v_i or v_j. This gives rise to three cases.

1. The support graph of a confluent flow could induce two trees on $G_{i,j}$ which must then be rooted at v_i and v_j. That is, flow is splitted into two components and some flow units leave $G_{i,j}$ at v_i and the other leave at v_j. We denote this case by $\mathrm{SPLIT}_{i,j}$. To compute the corresponding entries in the dynamic program, we can treat v_i and v_j as sinks in a smaller confluent flow instance consisting only of $G_{i,j}$. Since this instance would have two sinks, we may connect them to a supersink, at least in mind.

2. Otherwise, the tree of flow carrying arcs induces only one component in $G_{i,j}$ which is either rooted at v_i or v_j. If it is rooted at v_i, then we say the flow is routed left and treat v_i as a sink. This is the case $\mathrm{LEFT}_{i,j}$. But it may be that additional flow enters $G_{i,j}$ at v_j in an optimum solution. Therefore we have to treat v_j as a source with *undetermined supply*. For a dynamic program, we therefore have to know the flow value at v_i for all possible supplies of v_j.

3. In the other case, called $\mathrm{RIGHT}_{i,j}$, the flow leaves $G_{i,j}$ through v_j, which acts as sink, while v_i is a variable source.

The dynamic program needs a separate table for each of the three cases. For $\mathrm{SPLIT}_{i,j}$, all dominating pairs (in terms of Pareto maximal) of flow values arriving at v_i and v_j are computed. We will show that there can be only $(j-i)$ such values, as these situations can be characterized by a split point between v_i and v_j.

For $\mathrm{LEFT}_{i,j}$ (and analogously for $\mathrm{RIGHT}_{i,j}$), the desired table is a function describing the maximum flow that can reach the temporary sink v_i for each possible amount of supply at v_j, and not just for $d(v_j)$. Without further effort, these tables would have a pseudo-polynomial size. However, this function is what we want to call *simple*: For every unit of additional supply at v_j, the output at v_i can increase by 1 or remain unchanged. We will show that the number of times this behavior changes is bounded by $(j-i)^2$. If we additionally perform the entire computations with such a piecewise description of the tables in mind, we obtain a polynomial running time. Corresponding lemmata and proofs will follow in the next section.

To have a more symmetric notation, we actually want to split up the single uncapacitated sink v_0 into two sinks: We introduce a new node v_n between v_0 and v_{n-1} and make it a second sink. The capacities between v_{n-1} and v_n are inherited from the former arcs (v_{n-1}, v_0) and (v_0, v_{n-1}), but there is no arc added between v_0 and v_n. All the chords to and from v_0 remain at v_0. Both sinks again have infinite capacity. One can easily see that the original instance has the same Maximum Confluent Flow value as the new instance, because capacities did not change. We will further assume that these sinks use no outgoing arc because they have infinite demands. These preparations allows us to focus on the following kind of instances:

Definition 3.21 (Outerplanar instance). *An* outerplanar instance *consists of an outerplanar graph $G = (V, A)$ with the vertices $V = \{v_0, \dots, v_n\}$ embedded on the regular $(n+1)$-gon and labelled clockwise, arc capcities u, supplies d, and sinks v_0 and v_n with infinite demand. Furthermore, all arcs along the outer cycle except (v_0, v_n) and (v_n, v_0) must be in A.*

The following definitions formalize the dynamic programming tables.

Definition 3.22. *Consider an outerplanar instance and $i, j \in \mathbb{N}_0$ with $0 \leq i < j \leq n$.*

- *Let $G_{i,j} := G[\{v_i, v_{i+1}, \dots, v_j\}]$.*
- *Let $\mathrm{LEFT}_{i,j}(z) : \mathbb{N}_0 \to \mathbb{N}_0$ be the maximum value of a confluent S-T-flow in $G_{i,j}$ with capacities u, supplies d except $d(v_i) := 0$, $d(v_j) := z$, and $c(v_i) := \infty$.*
- *Let $\mathrm{LEFT}_{i,i}(z) := 0$ for all $z \in \mathbb{N}_0$.*
- *Let $\mathrm{RIGHT}_{i,j}(z) : \mathbb{N}_0 \to \mathbb{N}_0$ be the maximum value of a confluent S-T-flow in $G_{i,j}$ with capacities u, supplies d except $d(v_i) := z$, $d(v_j) := 0$ and $c(v_j) := \infty$.*
- *Let $\mathrm{RIGHT}_{i,i}(z) := 0$ for all $z \in \mathbb{N}_0$.*
- *Let $\mathrm{SPLIT}_{i,j} \subseteq \mathbb{N}_0^2$ contain exactly those pairs (l_k, r_k) with $l_k = \mathrm{LEFT}_{i,k}(d_k)$ and $r_k = \mathrm{RIGHT}_{k+1,j}(d_{k+1})$ for $k \in \{i, \dots, j-1\}$.*

Figure 15 shows an example why the RIGHT functions for larger subgraphs are not trivial. The crucial observation is that more flow entering on the left does not increase the output on the right immediately, but at some point the optimal solution might switch to an in-tree that allows for more capacity on the chord, but routes less of the flow originating inside $G_{i,j}$ to the right, still resulting in a net gain. In contrast, if the chord is not needed to route anything past $G_{i,j}$, then it is usually more effective to route the flow from vertices on the left through the chord instead of along the outer cycle. Finding the right balance between capacity on the chord and sending flow from inside to the outside is the trick. Note that this leads to optimal solutions that send flow in a *zigzag path* (see Figure 14) to the sink, which is quite unlike the flow in a cycle.

We now show the reasoning behind the definition of $\mathrm{SPLIT}_{i,j}$ and how we can compute the optimum Maximum Confluent Flow value from $\mathrm{SPLIT}_{0,n}$. The key property of the outerplanar embedding is that if we consider two undirected chords $\{v_i, v_j\}$ and $\{v_k, v_l\}$ with $i < j$ and $i < k < l$, these chords would cross if and only if $i < k < j < l$. This can be extended to paths: If there is a path from v_j to v_i (going left) and from v_l to v_k, then these paths must have a vertex in common if $i < k < j < l$. But neither chords in an outerplanar graph nor flow carrying paths in a confluent flow are crossing each other. Thus, there always is a vertex v_m with the highest index that sends flow to v_0 in an optimum solution and no flow is sent between $G_{0,m}$ and $G_{m+1,n}$. This yields the next lemma.

Lemma 3.23. *Consider an outerplanar instance. Let x be a confluent S-T-flow for this instance. W.l.o.g., assume v_0 and v_n use no outgoing arc. Then there exists some $m \in \{0, \dots, n-1\}$ such that all flow-carrying arcs F_x lie in the arc set $A(G_{0,m}) \cup A(G_{m+1,n})$.*

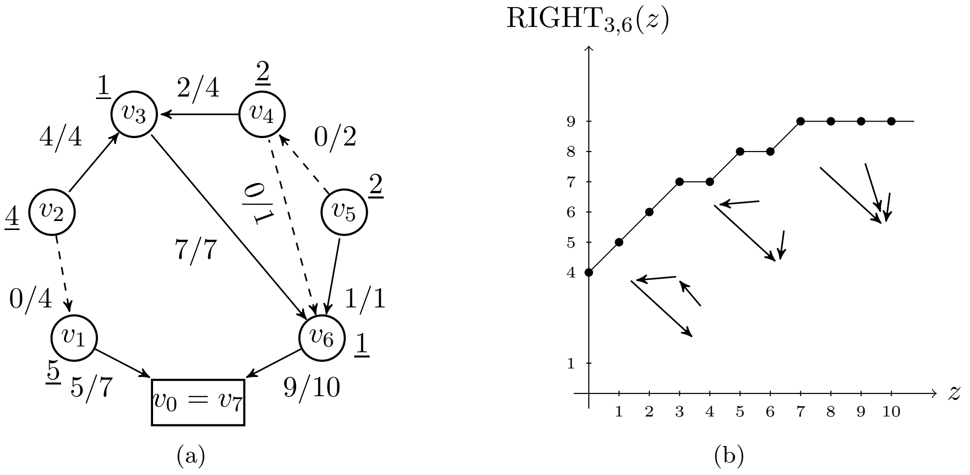

Figure 15: Confluent flow on a bagend: (a) The supplies d are underlined and the optimal solution is written as x/u on the arcs. (b) The plot of RIGHT$_{3,6}$ and the defining in-trees below.

Proof. For $i \in \{0, 1, \ldots, n\}$, let P_i be the path from v_i to either v_0 or v_n in F_x, or $P_i = \emptyset$ if there is no path from v_i to the sinks in F_x. Let $m := \max\{i : 0 \leq i \leq n-1 \wedge \emptyset \neq V(P_i) \subseteq \{v_0, v_1, \ldots, v_i\}\}$. Since $i = 0$ is always admissible, m is properly defined. We claim that this is the desired index.

Suppose there exists an arc in $F_x \setminus A(G_{0,m}) \cup A(G_{m+1,n})$. We need to treat the two directions of this arc slightly differently.

If there is an arc (v_i, v_j) from $G_{0,m}$ to $G_{m+1,n}$, then P_m cannot contain v_i, which would imply containing v_j as well, because P_m does not contain any vertex with a larger index than m. Then there must be a chord (v_k, v_l) in P_m with $l < i < k \leq m < j$ that enables P_m to bypass v_i. However, (v_k, v_l) would cross the chord (v_i, v_j).

Now that we have established that no chord carries flow from $G_{0,m}$ to $G_{m+1,n}$, we consider the case that there is a chord (v_j, v_i) carrying flow in the other direction. Since v_n has no outgoing flow, $j < n$ must hold. This implies $V(P_j) = V(P_i) \cup \{v_j\}$ and P_i cannot use any chord going back to $G_{m+1,n}$ as just shown. This makes $j > m$ eligible for the choice of m, a contradiction. □

Given SPLIT$_{0,n}$, it is now trivial to find the optimum Maximum Confluent Flow value.

Lemma 3.24. *The optimum* Maximum Confluent Flow *value of an outerplanar instance is*

$$\max_{(l,r) \in \mathrm{SPLIT}_{0,n}} (l + r).$$

Proof. Choose (l, r) that attains the maximum. By definition of SPLIT$_{0,n}$, this implies the existence of two confluent flows on the disjoint subgraphs $G_{0,k}$ and $G_{k+1,n}$ with sinks

v_0 and v_n, respectively, and with the original demands. Thus, we can easily combine them to obtain a feasible solution. (Note that for $k = 0$ or $k = n - 1$, one of the subgraphs consists of a single vertex and no arc. Thus, the flow is actually an empty function. This is not a problem, though, as $\text{LEFT}_{i,i}$ and $\text{RIGHT}_{i,i}$ are always 0.)

On the other hand, Lemma 3.23 shows that for any confluent S-T-flow x there always is an index m where the outerplanar graph is split by F_x. Plugging m into the definition of $\text{SPLIT}_{0,n}$, we see that the optimum solution gives an entry $(l^*, r^*) \in \text{SPLIT}_{0,n}$ with l^* and r^* being the amount of flow sent to the two sinks. Then $l + r \geq l^* + r^*$ must be true, but the latter is the optimum value. □

3.3.3 Dynamic programming

We will now show how one can compute the functions LEFT and RIGHT efficiently in a dynamic program, which immediately implies the computation for SPLIT as well. As mentioned above, we compute the values for $G_{i,j}$ from self-contained subgraphs $G_{i,m}$ and $G_{m,j}$. For this we need to ensure that this is always possible.

Lemma 3.25. *Consider an outerplanar instance. For $j - i \geq 2$, $0 \leq i < j \leq n$, there always exist $m \in \{i+1, \dots, j-1\}$ such that there is no arc in G between $\{v_i, \dots, v_{m-1}\}$ and $\{v_{m+1}, \dots, v_j\}$ besides maybe (v_i, v_j) and (v_j, v_i).*

Proof. Let $j - i \geq 2$. If v_i has no incoming arc from or outgoing arc to $\{v_{i+2}, \dots, v_{j-1}\}$, then simply choose $m := i + 1$. This already fulfills the statement of the lemma.

Otherwise let $m := \max\{m' : i < m' < j \wedge ((v_i, v_{m'}) \in A \vee (v_{m'}, v_i) \in A)\}$. This is now well-defined. Any arc between a vertex in $\{v_{i+1}, \dots, v_{m-1}\}$ and a vertex in $\{v_{m+1}, \dots, v_j\}$ would be a chord, because it bypasses v_m. Then such a chord would also cross the chord (i, m) or (m, i), which is a contradiction to the assumed embedding of G. Any other arc which the lemma forbids would have to be incident to v_i. Then it either contradicts the choice of m or is actually a chord between v_i and v_j, which is allowed. □

Lemma 3.26. *Consider an outerplanar instance. Let $0 \leq i < m < j \leq n$ be as in Lemma 3.25, that is, $G_{i,m}$ and $G_{m,j}$ are self-contained. Assume the chords (v_i, v_j) and (v_j, v_i) exist but maybe with capacity 0.*

1. *Then* $\text{LEFT}_{i,i+1}(z) = \min\{u(v_{i+1}, v_i), z\}$ *holds.*
2. *Then* $\text{RIGHT}_{i,i+1}(z) = \min\{u(v_i, v_{i+1}), z\}$ *holds.*
3. *Define*

$$\text{CHORDLEFT}_{i,j}(z) := \max_{(l,r) \in \text{SPLIT}_{i,j}} l + \min\{u(v_j, v_i), z + r\}.$$

Then

$$\begin{aligned}\text{LEFT}_{i,j}(z) = \max\{&\text{LEFT}_{i,m}\left(\text{LEFT}_{m,j}(z) + d(m)\right), \\ &\text{CHORDLEFT}_{i,j}(z)\}\end{aligned}$$

holds.

4. Define

$$\mathrm{CHORDRIGHT}_{i,j}(z) := \max_{(l,r)\in\mathrm{SPLIT}_{i,j}} r + \min\{u(v_i,v_j), z+l\}.$$

Then

$$\begin{aligned}\mathrm{RIGHT}_{i,j}(z) = \max\{&\mathrm{RIGHT}_{m,j}\left(\mathrm{RIGHT}_{i,m}(z)+d(m)\right),\\ &\mathrm{CHORDRIGHT}_{i,j}(z)\}\end{aligned}$$

holds.

Proof. 1. The value $\mathrm{LEFT}_{i,i+1}(z)$ is the maximum value of a confluent flow over the single arc (v_{i+1},v_i) with supplies $d(v_i)=-\infty$ and $d(v_{i+1})=z$. The claim follows obviously.

2. Similarly for $\mathrm{RIGHT}_{i,i+1}$.

3. For a fixed z, $\mathrm{LEFT}_{i,j}(z)$ is defined to be the maximum confluent flow value in $G_{i,j}$ that can leave v_i if the supply of v_j is z and the intermediate vertices have their usual supply, but v_i has no supply of its own. Let x be such an optimal confluent flow and consider F_x. Since the single sink cannot use an outgoing arc, $(v_i,v_j)\notin F_x$ holds. First consider the case $(v_j,v_i)\notin F_x$: Then any flow originating in $\{v_m,\dots,v_j\}$ must be routed through v_m to reach v_i, by the choice of m. Thus, we can see this as additional supply on v_m. The maximum supply available at v_m is then $\mathrm{LEFT}_{m,j}(z)+d(m)$. So the maximum flow reaching v_i if $(v_j,v_i)\notin F_x$ is $\mathrm{LEFT}_{i,m}(\mathrm{LEFT}_{m,j}(z)+d(m))$, which constitutes the first part of the maximum that defines $\mathrm{LEFT}_{i,j}(z)$.

   Now suppose that $(v_j,v_i)\in F_x$. Then x can be interpreted as a flow on $G_{i,j}$ but with sinks v_i and v_j and ignoring the flow on the chord. (It may not be the optimal confluent flow for this instance, though.) We can apply Lemma 3.23 to this smaller instance, yielding a $k\in\{i,\dots,j-1\}$ such that the two trees pointing towards the sinks in F_x are contained in $\{v_i,\dots,v_k\}$ and $\{v_{k+1},\dots,v_j\}$, respectively. Then the flow value of x in this new instance is at most $\mathrm{LEFT}_{i,k}(d_k)+\mathrm{RIGHT}_{k+1,j}(d_{k+1})$. Note that if $k=i$ or $k=j$, the special case definitions of $\mathrm{LEFT}_{i,i}\equiv 0$ and $\mathrm{RIGHT}_{j,j}\equiv 0$ ensure that these sinks by themselves do not contribute any flow value.

   Since v_j is not really a sink, but has supply z and uses the chord (v_j,v_i) to send to v_i, the flow value of the contribution from v_j is increased by z and bounded by $u(v_j,v_i)$. So assuming x is the optimal confluent flow, we then obtain $\mathrm{LEFT}_{i,j}(z)=\mathrm{LEFT}_{i,k}(d_k)+\min\{u(v_j,v_i),\mathrm{RIGHT}_{k+1,j}(d_{k+1})+z\}$.

   Since we do not know if (v_j,v_i) is used (if it is available at all) and which vertex v_k is the splitting point, we have to try all combinations and take the maximum value. The function $\mathrm{CHORDLEFT}_{i,j}(z)$ summarizes all cases where $(v_j,v_i)\in F_x$, and is therefore the only other option that needs to be considered for the maximum defining $\mathrm{LEFT}_{i,j}(z)$.

4. As for $\mathrm{LEFT}_{i,j}$.

$\square$

3.3.4 Simple functions

Using the formulae developed in the previous section to evaluate the functions at certain values of z is not advisable in a polynomial algorithm. Computing $\mathrm{LEFT}_{i,j}$ or $\mathrm{RIGHT}_{i,j}$ in the interesting case where there is a chord (i, j) or (j, i) with positive capacity relies on $\mathrm{SPLIT}_{i,j}$. This again relies on LEFT and RIGHT evaluations for $j - i$ subgraphs, which can be computed recursively. In particular, the same pair of indices might occur in different subtrees of the recursion. So we want to use a dynamic programming approach instead, which computes the required values in a bottom-up way. There is one big problem, though, compared to the recursion: We do not know for which values of z we need to evaluate the LEFT and RIGHT functions, so we will compute the *entire* functions! Done naïvely, this only yields a pseudo-polynomial algorithm because z may take pseudo-polynomially many values. However, we can *describe* these functions efficiently, making use of their simple "stair-case" form as seen in Figure 15(b). Even better, we can carry out the *computations* in Lemma 3.26 on these efficient descriptions much faster than for all values of z by themselves.

Definition 3.27 (Simple function). *Let $f : \mathbb{N}_0 \to \mathbb{N}_0$ be a function such that $f(z+1) \in \{f(z), f(z) + 1\}$ for all $z \in \mathbb{N}_0$. Then we call f a* simple *function. Such a function can be described by $f(0)$ and the (inclusion maximal) intervals on which f increases. We denote the size of this set of intervals by $\langle f \rangle$.*

For example, for the simple function in Figure 15(b) we store $f(0) = 4$ and the set of increasing intervals $\{[0, 3], [4, 5], [6, 7]\}$. Note that it is easy to store the $\langle f \rangle$ intervals describing f in a linked list. This trivially allows the evaluation of $f(z)$ in $\mathcal{O}(\langle f \rangle)$.

Lemma 3.28. $\mathrm{LEFT}_{i,j}$ *and* $\mathrm{RIGHT}_{i,j}$ *are simple.*

Proof. As $\mathrm{LEFT}_{i,j}$ describes a maximum flow value, $\mathrm{LEFT}_{i,j}(z + 1) \geq \mathrm{LEFT}_{i,j}(z)$, because the flow cannot become worse with higher supply by using the same in-forest F_x of flow carrying arcs. Also $\mathrm{LEFT}_{i,j}(z+1) \leq \mathrm{LEFT}_{i,j}(z)+1$ holds, because any in-forest used for $\mathrm{LEFT}_{i,j}(z+1)$ could have been used for $\mathrm{LEFT}_{i,j}(z)$, losing at most one unit of flow. $\mathrm{RIGHT}_{i,j}$ is simple by the same arguments. $\square$

We can operate on simple functions as follows:

Lemma 3.29. *Let f_1 and f_2 be two simple functions.*

1. *$g(z) := f_1(f_2(z))$ is simple and $\langle g \rangle \leq \langle f_1 \rangle + \langle f_2 \rangle$.*
2. *$g(z) := \max\{f_1(z), f_2(z)\}$ is simple and $\langle g \rangle \leq \langle f_1 \rangle + \langle f_2 \rangle$.*
3. *For $c \in \mathbb{N}_0$, $g(z) := f_1(z) + c$ is simple and $\langle g \rangle = \langle f_1 \rangle$.*
4. *For $c \in \mathbb{N}_0$, $g(z) := f_1(z + c)$ is simple and $\langle g \rangle \leq \langle f_1 \rangle$.*

In all cases, g can be computed in time linear in $\langle f_1\rangle$ and $\langle f_2\rangle$, where applicable.

Proof. 1. The definitions immediately imply $f_1(f_2(z+1)) \leq f_1(f_2(z)+1) \leq f_1(f_2(z))+1$ and $f_1(f_2(z+1)) \geq f_1(f_2(z))$. So, $g(z)$ is simple. Whenever $f_1(f_2(z))$ changes between increasing and constant behavior there has to be a change in the behavior of f_1 or f_2. Therefore $f_1(f_2(z))$ can have at most $\langle f_1\rangle + \langle f_2\rangle$ intervals on which it increases. The computation can be carried out by a scan through the interval list of f_2, pushing another pointer in f_1 forward whenever $f_2(z)$ has increased enough.

2. $g(z+1) \geq f_1(z+1) \geq f_1(z)$ and $g(z+1) \geq f_2(z+1) \geq f_2(z)$, so $g(z+1) \geq \max\{f_1(z), f_1(z)\} = g(z)$. On the other hand, if $g(z+1) = f_1(z+1)$, then $g(z+1) \leq f_1(z)+1 \leq g(z)+1$, and analogously for the case $g(z+1) = f_2(z+1)$. So, $g(z)$ is simple. Any increasing interval of $g(z)$ must be contained in an increasing interval of $f_1(z)$ or $f_2(z)$. However, the same increasing interval of $f_1(z)$ or $f_2(z)$ cannot cause more than one increasing interval in $g(z)$: If it did, the endpoints of the subintervals would still have the same values and distances as in the original function, so the slope in between must always be 1 for $g(z)$, meaning that the increasing interval in $g(z)$ was uninterrupted.
The computation is a scan along both functions simultaneously.

3. This only shifts the constant $f_1(0)$.

4. This shifts the entire function to the left by c. All intervals ending before c have to be deleted, and $f_1(0)$ adjusted accordingly.

□

We can now apply these considerations to the recursive definitions in Lemma 3.26.

Lemma 3.30. *For all $0 \leq i \leq j \leq n$ the following holds:*

1. $|\text{SPLIT}_{i,j}| \leq j-i$ *(assuming $i<j$).*

2. $\langle \text{LEFT}_{i,j}\rangle \leq (j-i)^2$.

3. $\langle \text{RIGHT}_{i,j}\rangle \leq (j-i)^2$.

Proof. 1. All the entries in $\text{SPLIT}_{i,j}$ are determined by one of the values of $k \in \{i,\dots,j-1\}$.

2. We use induction on $j-i$. For $j-i=0$, the claim holds because $\text{LEFT}_{i,i} \equiv 0$ and $\langle \text{LEFT}_{i,i}\rangle = 0$. Similarly elemental, for $j-i=1$, we note $\langle \min\{u_{v_j,v_i}, z\}\rangle \leq 1$.

   Induction and Lemma 3.29 yield $\langle \text{LEFT}_{i,m}(\text{LEFT}_{m,j}(z)+d(m))\rangle \leq (j-m)^2 + (m-i)^2$.

   Now consider $\text{CHORDLEFT}_{i,j}$. It is the maximum over $|\text{SPLIT}_{i,j}| \leq j-i$ simple functions of size at most 1. So $\langle \text{CHORDLEFT}_{i,j}\rangle \leq j-i$. This plus the size of $\text{LEFT}_{i,m}(\text{LEFT}_{m,j}(z)+d(m))$ bounds the size of $\text{LEFT}_{i,j}$ due to the computation by a maximum over the two. But $(j-m)^2+(m-i)^2+j-i \leq (j-m)^2+(m-i)^2+2(j-m)(m-i)$ because $i<m<j$, and the latter is simply $(j-i)^2$.

3. As above.

□

Now all the pieces for a polynomial algorithm for Maximum Confluent Flow on outerplanar graphs with one sink are set up.

Theorem 3.31. Maximum Confluent Flow *on an outerplanar graph G with a single sink can be solved in polynomial time.*

Proof. First, find an embedding of G and transform the input into an outerplanar instance. Then, starting with $j - i = 0$ and proceeding towards $j - i = n$, compute all $\text{SPLIT}_{i,j}$, then $\text{LEFT}_{i,j}$ and $\text{RIGHT}_{i,j}$ inductively in a dynamic program. For each of these pairs (i, j), $\text{SPLIT}_{i,j}$ can be computed from various $\text{LEFT}_{i,k}$ and $\text{RIGHT}_{k+1,j}$, with $k - i < j - i$ and $j - (k + 1) < j - i$. Having done so, compute $\text{LEFT}_{i,j}$ from $\text{SPLIT}_{i,j}$, $\text{LEFT}_{i,m}$ and $\text{LEFT}_{m,j}$ with $i < m < j$, and similarly for $\text{RIGHT}_{i,j}$. Importantly, all of these calculations are performed on the interval representation of the simple functions. □

3.3.5 Runtime analysis

Lemma 3.30 shows that at no time the size of SPLIT can be larger than $j - i$, while LEFT and RIGHT need at most $(j - i)^2$ entries to describe them. Computing each $\text{SPLIT}_{i,j}$ requires $\mathcal{O}(n)$ evaluations of LEFT and RIGHT functions, each of which can be done in $\mathcal{O}(\langle\text{LEFT}\rangle) = \mathcal{O}(n^2)$. So for each $\text{SPLIT}_{i,j}$ we need at most time $\mathcal{O}(n^3)$. Computing LEFT requires 2 evaluations of LEFT and the maximum over $\mathcal{O}(n)$ entries in SPLIT. So this takes $\mathcal{O}(n^2)$ time. Note that we can find the splitting point m with a simple greedy strategy by checking the existence of $\mathcal{O}(n)$ arcs, as shown the proof of Lemma 3.26. Clearly, this is dominated by $\mathcal{O}(n^3)$. So for each pair (i, j) we can find all necessary functions in $\mathcal{O}(n^3)$, and the entire table is computed in $\mathcal{O}(n^5)$. Finally, the maximum value can be found in $\mathcal{O}(n)$ from $\text{SPLIT}_{0,n}$, as noted in Lemma 3.24.

While the running time might not seem too good, here are a few things to consider: So far, we do not know of any instance, where LEFT and RIGHT actually have more than linearly many intervals. Maybe one can tighten the analysis of Lemma 3.30, which would immediately imply a faster running time. Furthermore, a better algorithm is already possible because not all pairs (i, j) are actually required to compute $\text{SPLIT}_{0,n}$. Indeed, the costly computation of $\text{SPLIT}_{i,j}$ is only needed if there is a chord (v_i, v_j) or (v_j, v_i) because otherwise CHORDLEFT is not a good option. But there are only $\mathcal{O}(n)$ chords in an outerplanar graph (actually, any planar graph has only $\mathcal{O}(n)$ edges), so an improved algorithm would need to construct $\text{SPLIT}_{i,j}$ only for $\mathcal{O}(n)$ pairs of indices. To keep it simple, we ignored this matter of fact in the construction of our algorithm, but a running time of $\mathcal{O}(n^4)$ could already be realized by a proper implementation.

3.3.6 Confluent flows and minors

A minor M of an undirected graph G is a graph that can be obtained out of a subgraph of G by zero or more edge contractions. Outerplanar graphs are exactly those graphs that do not contain a K_4 nor a $K_{2,3}$ as a minor [27]. They are in particular series-parallel graphs, which are exactly those graphs that are K_4-free [24]. But we cannot hope to solve Maximum Confluent Flow on this larger class of K_4-free graphs in polynomial time because the $\mathcal{NP}$-hard problem with treewidth 2 (see Section 2.4, Figure 4) is series-parallel. In this section we will try to identify the $K_{2,n}$ as the key minor for the complexity of confluent flows. We start with the following lemmata.

Lemma 3.32. *Let G be a (undirected) graph which does not contain the complete bipartite graph $K_{2,3}$ as a minor. If G contains the complete graph K_4 as a minor, then the K_4 is an induced subgraph G.*

Proof. Assume, that G contains the K_4 as minor, but G is K_4-free. Thus, there is a sequence of edge contractions on a subgraph G' of G, which lead to the K_4. Let $(G', G_1, G_2, \ldots, G_k = K_4)$ be the sequence of the resulting graphs. There exists a minimum index i such that G_i contains a K_4 as induced subgraph. W.l.o.g. $i = k$, i.e. the K_4 is created in the last step for the first time. All redundant nodes and vertices can be deleted before. G_{k-1} has 5 nodes and at least 7 edges (since the resulting K_4 has 4 nodes and 6 edges). Thus, G_{k-1} is a K_5 with up to three deleted edges. These three missing edges cannot be incident to the same node, because this implies a K_4 on the other 4 nodes. With the same argument, at least two edges have to be missing. On the other hand, 6 vertices are involved in three edges. Hence, at least one vertex is incident to at least two missing edges by the pigeonhole principle in this case.

Now, it is easy to check the remaining cases for G_{k-1}. In fact, there is only one case where G_{k-1} contains only 7 edges and can be transformed to a K_4. The graph and its induced $K_{2,3}$ are presented in Figure 16. Each case with 8 edges also contains a induced $K_{2,3}$. Hence, we have a contradiction.

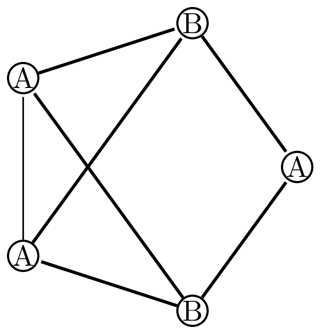

Figure 16: Minimum example for a graph which contains a K_4 as a minor but not as an induced subgraph. Nevertheless, it contains a $K_{2,3}$ as an induced subgraph. The both partitions of the bipartite subgraph are labeled with A and B.

□

Lemma 3.33. *Let G be a (undirected) graph which does not contain the complete bipartite graph $K_{2,3}$ as a minor. Furthermore, let G contain the K_4 as an induced subgraph.*

Let v and w be two arbitrary vertices of this induced K_4, then there is no path between v and w which contains any other vertices than the nodes of the K_4.

Proof. Assume there exists a path P which contains a vertex not belonging to the K_4. W.l.o.g. P starts at v, ends at w and contains no other vertices of the K_4. Delete all nodes and edges which do not belong to the K_4 or P. Contract the edges of P until only two edges remains. The resulting graph has a $K_{2,3}$ as an induced subgraph (Figure 17), a contradiction.

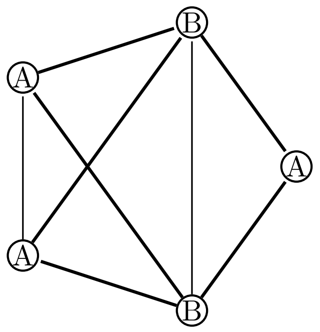

Figure 17: Two nodes of a K_4 connected via a path always induces a $K_{2,3}$.

□

With these two lemmata, it is easy to understand the structure of a graph that contains no $K_{2,3}$ as a minor. The graph may contain some induced K_4 subgraphs. Deleting all edges of these K_4 the graph decomposes in some outerplanar components. Lemma 3.33 ensures that two induced K_4 or an induced K_4 and an induced outerplanar subgraph have at most one common vertex. Furthermore, it guarantees a tree structure of these components, since there are no paths connecting two nodes of a K_4. The underlying tree H can be constructed easily. For each induced K_4 and each induced maximal outerplanar component add one vertex. Connect two nodes if and only if the corresponding components have a common vertex. H is similar to the block graph of G. Note that two nodes corresponding to outerplanar components are never connected directly since we consider only maximal outerplanar components. Thus, each cycle in H would contain at least one node corresponding to a K_4. Therefore, a cycle would induce a forbidden path between two nodes of the K_4. Hence, H contains no cycles. We can use this underlying tree structure to solve Maximum Confluent Flow on graphs without $K_{2,3}$ minor.

Corollary 3.34. *If G contains no $K_{2,3}$ as a minor, then* Maximum Confluent Flow *with a single sink can be solved in polynomial time.*

Proof. We exploit the underlying tree structure. First, construct H by decomposing G into maximal outerplanar components and K_4 components. H is a tree, thus the direction of flow is well defined on H and therefore on each component. W.l.o.g. the sink is contained in only one component, which corresponds to the root of H (add a new sink and connect it to the old one via one arc). The components have at most one common vertex. These vertices act as temporary sources or sinks and we compute a maximum confluent S-T-flow for each component starting at the leaves of H. The incoming flow of a temporary sink defines the maximum demand of the corresponding source in the parent component. This implies that the maximum value of a confluent flow in G is equal to the maximum value of a confluent flow in the root component. Maximum values of confluent flows on outerplanar components are processed as in Theorem 3.31. K_4 components have fixed size, therefore a maximum confluent flow can be found in $\mathcal{O}(1)$. Obviously, the number of components is bounded by the size of G. Thus, the claim follows. □

Note that there is no polynomial time algorithm for the class of $K_{3,3}$-free graphs (unless $\mathcal{P} = \mathcal{NP}$), as this allows a $K_{2,k}$, which can again model a PARTITION instance (cf. Section 2.4, Figure 4). However, there is room for one stronger conjecture:

Conjecture 3.35. *For any constant $k \in \mathbb{N}$,* MAXIMUM CONFLUENT FLOW *on $K_{2,k}$-free graphs can be solved in polynomial time.*

A graph that is $K_{2,k}$-free is probably too restrictive to contain arbitrarily large $\mathcal{NP}$-hard subproblems like PARTITION. In particular, this weaker requirement could also allow outerplanar graphs with up to $k - 1$ sinks modeled by a supersink. However, an algorithm for this problem might be quite involved.

3.4 Planar graphs with all terminals on the boundary

3.4.1 Boundary instances

The class of outerplanar graphs is a very restricted graph class. In this section, we relax the restrictions by allowing some additional nodes and arcs inside the cycle. In other words, we consider planar graphs with all terminals on a common face. For clearness, we choose an embedding such that this face is the outer face, i.e. all sources and the sink are placed on the *boundary*. Unfortunately, the results in this section do not hold for an arbitrary number of sources. Instead, we always have to assume the number of sources to be bounded by a constant k.

This section is motivated by related disjoint path problems. Okamura and Seymour [123] studied edge disjoint paths in the above-mentioned setting. They proved an existence theorem under a certain evenness condition (cf. Section 2.3.3). Several fast algorithms for edge disjoint paths in this so-called *Okamura-Seymour-case* were developed [13, 86, 148]. Furthermore, this was also generalized to multicommodity flows on such graphs, e.g. [151].

Definition 3.36 (Boundary instance). *A* boundary instance *consists of an embedded planar graph $G = (V, A)$, where the set of k sources $\{s_1, \ldots, s_k\}$ and a single sink t lie on the boundary of this embedding. The sources are labeled counterclockwise such that t is placed between s_k and s_1. Furthermore, arc capacities u, infinite supplies d at the sources, and an infinite sink capacity are given.*

For brevity, we use infinite demands here. This is no restriction for Maximum Confluent Flow. As always, limited demands can be simulated by adding new sources with infinite demand and arcs with the desired capacity ahead of the original sources.

Again, we start with a high-level description. The algorithm is based on *recursive bottleneck path* computations. However, we cannot take the first bottleneck path that comes along, since a bad choice may block other paths.

At this point, we will exploit the special properties of the boundary instance. It provides an orientation between paths, i.e. we can define whether a path is *left* or *right* of another path. This also defines a partial ordering of arbitrary paths. For paths of a confluent flow which must not cross, we obtain a total ordering. Therefore, for a certain source s_l and a capacity z, we will define a *left most transshipment*, i.e. the path from s_l to t that is left of every other path among all s_l-t-paths which provide a capacity of at least z flow units.

Furthermore, Algorithm 3.1 for trees tells us that we can process the sources in any order, when the in-tree is given. Consequently, we will process them in their natural ordering, that is from s_1 to s_k. We will start with exploring all critical left most transshipments from s_1 to t. This leaves a residual network for each path and we will continue with the next source recursively.

Lemma 3.37. *Given two paths from s_i to t and from s_j to t in a boundary instance, let $i < j$. If both paths meet in a node v in a confluent flow, then flow from s_l, $i < l < j$ also passes through v.*

Proof. The claim is obvious. Since the graph is planar and all terminals are on the boundary, there is no possibility that any paths bypass each other. ☐

3.4.2 Preliminaries

Let us start with the definitions for *left of* and *left most transshipment*. On the one hand, left and right are very intuitive. On the other hand, a robust definition is rather technical. Figure 18 explains left and right in a simple way.

Given a boundary instance with sources ordered counterclockwise and a path P from s_l to t, we add the arc (t, s_l) crossing the outer face. Now, we consider the dual graph $\bar{G}$ of $G = (V, A \cup \{(t, s_l)\})$, i.e. each face of the graph is identified by a vertex and two vertices are connected if the corresponding faces share a common arc. Let $\bar{v}_l$ of the dual graph be the node corresponding to the face adjacent to t and $s_1, \ldots, s_l$. Similarly, $\bar{v}_r$ is the node corresponding to the face adjacent to t and $s_l, \ldots, s_k$. The cycle $P \cup (t, s_l)$ defines a cut in the dual graph. Furthermore, it splits $\bar{G}$ in exactly two connected components. Let $\bar{V}_l$ be the node set of the component which contains $\bar{v}_l$ and $\bar{V}_r$ is the node set of the component which contains $\bar{v}_r$. Now, $V_l \subseteq V$ is the node set of all vertices adjacent to faces which correspond to nodes in $\bar{V}_l$. V_l is the *node set left of* P. Similarly, V_r is defined as the *node set right of* P. $V_l \cap V_r$ is exactly the node set visited by P.

Definition 3.38 (Left of, right of). *Given a boundary instance and a path P from s_i to t. P separates the vertices into two sets V_l and V_r as above. A path P' from s_j is* left

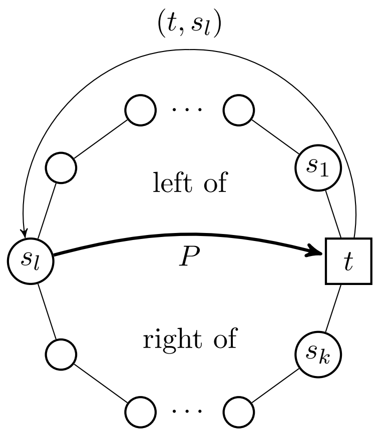

Figure 18: A path P in a boundary instance with the node sets left and right of P.

of P *if it only visits vertices of* V_l. *Similarly, a path is* right of P, *if it only visits nodes of* V_r.

Obviously, there exist paths that are neither left nor right of P. Thus, *left of* defines a *partial ordering.* Note that this relation is reflexive, i.e. each path is left of itself.

Definition 3.39 (Left most transshipment). *Given a boundary instance with sink t, a specified source s and a value z, a* left most transshipment *$lmt(s, z)$ is a path $P_{s,z}$ from s to t and a flow on this path with value equal to z, such that* $z \leq \min_{a \in P_{s,z}} u(a)$ *and* $z > \min_{a \in P'} u(a)$ *for all paths P' left of $P_{s,z}$.*

Obviously, a left most transshipment is well-defined, i.e. it is unique. Furthermore, it is easy to compute if the embedding is given. If we have found such a path we may also use it at full capacity, that is its bottleneck capacity. The following definition increases the flow value of a left most transshipment up to its bottleneck.

Definition 3.40 (Left most flow). *Given a boundary instance with sink t, a specified source s and a value z, a* left most flow *$lmf(s, z)$ is a path $P_{s,\bar{z}}$ from s to t and a flow with flow value equal to $\bar{z}$ on this path, such that* $z \leq \bar{z} = \min_{a \in P_{s,\bar{z}}} u(a)$ *and* $z > \min_{a \in P'} u(a)$ *for all paths P' left of $P_{s,\bar{z}}$.*

The intention of a left most flow is the following. Assume we want to send at least z units of flow from s_1. We should send these flow units in a way such that we do not block confluent flow from other sources. $P_{s_1,\bar{z}}$ seems to be a good choice. However, this is not completely true.

On the one hand, $P_{s_1,\bar{z}}$ is optimal before it reaches its bottleneck. We cannot use any other path left of $P_{s_1,\bar{z}}$, because there is none. Thinking of $P_{s_1,\bar{z}}$ as part of an in-tree, it is used at full capacity, which is also desired. Furthermore, there is no use that additional flow from other sources is routed to this first part of the path, since the bottleneck is still ahead. On the other hand, it is easy to see that $P_{s_1,\bar{z}}$ can be a bad choice after its bottleneck. Maybe, flow from s_2 has to meet $P_{s_1,\bar{z}}$ directly after the bottleneck arc. Now, the confluent condition forces this flow to follow $P_{s_1,\bar{z}}$ which may limit the additional capacity drastically (cf. Figure 19).

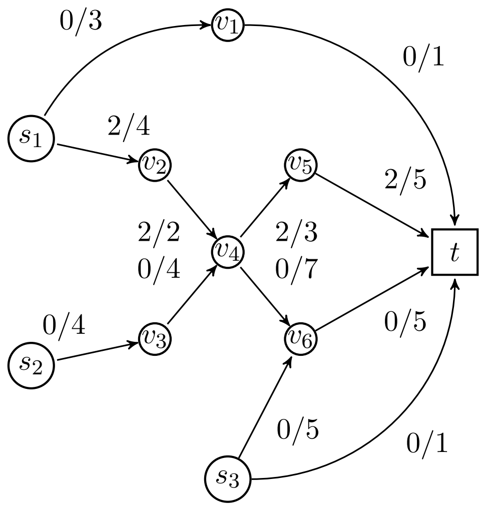

Figure 19: Boundary instance with a left most transshipment of 2 flow units from s_1 to t. The arc labels denote arc flow and arc capacity in the form *flow/capacity*. Assume, we are going to send flow from s_2 to t. There is a path which provides a capacity of 4. Unfortunately, this path meets the previous path at v_4, thus, we have to pick up these two flow units. Consequently, we may only send 3 flow units from s_2 via v_3, v_4, and v_6 to t. However, we should also consider sending only one additional flow unit via v_5, since this may provide additional capacity for flow originating at s_3.

Nevertheless, we maintain the idea of a left most flow for the remaining section. For a source s_l, we will first choose the flow value that should be shipped at least. Afterwards, we search for a path on the left side of the network, which provides enough capacity. Finally, we increase flow on this path up to its bottleneck capacity and continue with the next source s_{l+1}.

However, if we meet the path from the previous source s_{l-1}, then we will usually have to re-route flow. That is, we should look for a new left most flow with respect to the sum of the flow values starting at the meeting point. We prepare this computation by defining a *residual network*.

Definition 3.41 (Residual network). *A* residual network *is a boundary instance with an additional label $q : V \to \mathbb{N}_0$. For the original instance, $q \equiv 0$.*

The label is used to keep track how much flow already passes a certain node v. The labels can also be seen as a temporary storage. If a path from another source meets this node v, both the stored flow and the new flow have to be routed together to the sink.

Otherwise, we would break the confluent flow condition in v. For small amounts of flow, the previous s_{l-1}-t-path may provide enough capacity such that the new path P can simply follow the previous path. For higher flow values, we have to re-route the stored flow $q(v)$.

For determining the capacity of such a new path P which meets a node with label greater than 0, the capacities of the remaining arcs have to be reduced by the amount of collected stored flow units. Assume, all sources from s_1 to s_{l-1} are already processed and we are going to send z flow units along a s_l-t-path $P = ((v_1, v_2), \dots, (v_{r-1}, v_r))$, i.e. $s = v_1$ and $t = v_r$. First, we have to check, whether this path provides enough capacity for sending z units of flow. This is true, if $u((v_j, v_{j+1})) \geq z + \max_{1 \leq i \leq j} q(v_i)$ $\forall j = 1, \dots, r-1$. Note that labels are not additive, since we may follow the previous path or meet it twice, thus, a label indicating the same flow units may appear several times. Afterwards, we use P at its bottleneck capacity $\bar{z} > z$ and update the residual network as follows.

Procedure 3.42 (Update of the residual network). *If flow is sent on a path $P = ((v_1, v_2), \dots, (v_{r-1}, v_r))$ with bottleneck capacity $\bar{z}$, then the residual instance is updated as follows:*

1. *Delete all nodes and incident arcs of $V \backslash V_r$, that is all nodes left of P besides the path itself.*
2. *Update the labels along P: $q(v_j) = q'(v_j) + \bar{z}$ with $q'(v_j) = \max_{1 \leq i \leq j} q(v_i)$.*
3. *Delete all outgoing arcs of all vertices that P visits before it reaches its last bottleneck arc.*
4. *Delete all remaining nodes that are in the same connected component as the sink.*

The embedding of the remaining nodes and arcs is not changed.

This procedure is reasonable. Since we are going to process the sources in ascending order, the left part of the network in the residual instance contains no more sources. Furthermore, the confluent flow condition permits crossing the path P. Thus, the left side of the network is not accessible for the remaining sources and the first update rule deletes it. The second update rule collects all flow values stored along P and propagates them. Thus, the labels are set to the current confluent flow that arrives at these nodes. Update rule 3 ensures the confluent flow condition on the first part of P before its bottleneck arc by deleting all alternative arcs. Note that each node with label greater than 0 is on the boundary of the updated residual instance. Furthermore, starting next to the sink, the labels are decreasing in counterclockwise direction.

Summarizing, the labels provide an interface to the current flow pattern of flow sent from $s_1, \dots, s_l$ to t in the part of the network which was already deleted. Originally, the flow stored in the labels was sent to the sink via these paths, it is equivalent to the current outflow of the nodes along the s_l-t-path. With help of the labels, we keep track of the flow for re-routing purposes.

Now, we are prepared to discuss the choice of the next flow-carrying path P in detail. Assume we have already processed all sources from s_1 to s_l, we proceed with s_{l+1} and we are going to send z flow units. Following the mantra of this section we try to stay as left as possible. If the new flow-carrying path P arrives at a node v on the boundary with label greater than 0, then it has to carry along this stored flow for the remaining way to the sink or it would break the confluent condition. Unfortunately, a path providing enough capacity for the summed flow value may imply a long detour on the right side of the instance. Consequently, it could be better to avoid picking up flow in this node v.

Further, we cannot directly apply the *left of* relation if the flow value changes on a path. Assume, we may choose between two possibilities P' and P'' for the next path. P' is a path leading directly to the sink t without hitting the boundary, its bottleneck capacity is z'. On path P'' we are going to send only $z'' < z'$ flow units. Thus, P'' can start left of P', but it may arrive at a node v on the boundary. If $z'' + q(v) > z'$, then the re-routing could force the P'' to the right side of P'. An example is depicted in Figure 20.

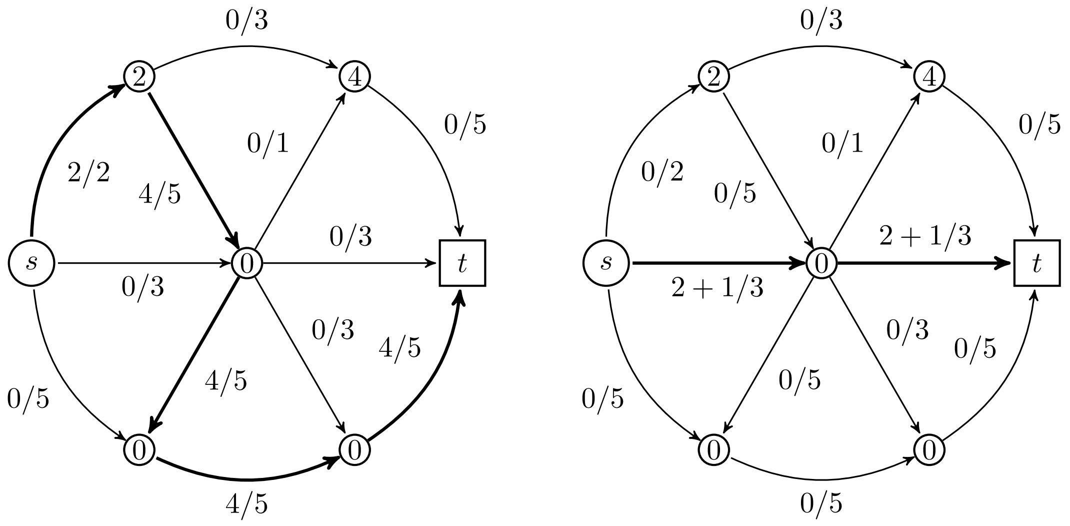

Figure 20: Assume we are trying to send 2 units of flow from s to t in the residual instance. The node labels are depicted inside the nodes. The arc labels denote arc flow and arc capacity as in Figure 19. On the left side, the 2 flow units are sent to a node on the boundary, where another 2 flow units are stored. Now, 4 units have to be shipped and the capacity constraints force the path to the right side. In the residual instance on the right side, the path also stays as left as possible, but it avoids the boundary except for t. Furthermore, this path provides capacity for one more flow unit. Thus, it is very important, where the new path hits the boundary.

We will use a workaround to avoid these difficulties. The update procedure has deleted all nodes left of the s_l-t-path, thus, only nodes on the boundary between t and s_l have a label with value greater than 0. We number the nodes on the boundary with a label greater than 0 in counterclockwise order from $w_0 = t$ to w_h. Hence, the index also provides a distance measure along the boundary. Furthermore, we set $w_\infty := s_{l+1}$.

Given a path P, it may visit several nodes w_i. We use $P^{i,j}$ to denote the part of P between w_i and w_j which must not visit any other vertex w_r. Thus, there is a unique decomposition of P into paths $P^{\infty,r_i} \circ P^{r_i,r_{i-1}} \circ \cdots \circ P^{r_1,0}$. Now, we consider paths, which use $P^{\infty,r}$ as first part for an arbitrary $r \in \{0, \dots, h-1\}$.

Definition 3.43 (Left most flow in a residual network). *Given a residual network with a specified source $s = w_\infty$ and nodes with label greater than 0 on the boundary labeled counterclockwise from $w_0 = t$ to w_h, a value $z \in \mathbb{R}_{\geq 0}$, and a number $r \in \mathbb{N}_0$, a* left most flow in the residual network *lmf*(s, z, r) *is a path $P = P^{\infty,r_i} \circ P^{r_i,r_{i-1}} \circ \cdots \circ P^{r_1,0}$ and a flow on this path with flow value equal to $\bar{z}$ such that*

1. *$\bar{z} \geq z$ is the remaining capacity of P with respect to the capacities in the residual instance,*
2. *P does not visit w_j for any $j > r$, i.e. $r_i = r$,*
3. *$P^{r_j,r_{j-1}}$ is the path of a left most transshipment from w_{r_j} to $w_{r_{j-1}}$ with flow value $z + q(w_j)$ $\forall j \in \{1, \dots, i\}$.*

In easier words, we are looking for a path from s to t which provides capacity z. We strictly avoid the nodes with stored flow on the boundary near to the source, but we have to visit w_r. Whenever we visit a node on the boundary, we pick up the flow stored in the label. For the remaining journey, we try to stay as left as possible until we visit another node on the boundary. This influence of the parameter r is depicted in Figure 21.

Please note that such a left most flow in a residual network may not exist, even if there is a path with capacity at least z. If we pick up too much flow in the boundary nodes, we may not find a way without exceeding an arc capacity. Nevertheless, computing a left most flow in the residual network for given values z and r is very easy, e.g. by a simple depth-first-search strategy using the embedding. At each node, the left most outgoing arc with respect to the incoming arc is tried first. We do not use an arc, if it does not provide enough capacity or if it leads to a forbidden node on the boundary.

3.4.3 An algorithm for boundary instances

Now, we can give an algorithm for Maximum Confluent Flow on boundary instances. The (planar) graph contains m arcs, thus, there are at most $\mathcal{O}(m) = \mathcal{O}(n)$ reasonable choices for bottleneck capacities $\bar{z}$. These capacities are reduced by at most k different values of flow on previous paths. Algorithm 3.6 simply enumerates all $\mathcal{O}(kn)$ possibilities of flow values sent out by the k sources in the residual network. Furthermore, there are at most n nodes on the boundary and the algorithm enumerates all choices of r for the left most flow in the residual network. That is, we use *brute force* to guess the optimal values of z and r for each source.

Lemma 3.44. *Algorithm 3.6 is correct.*

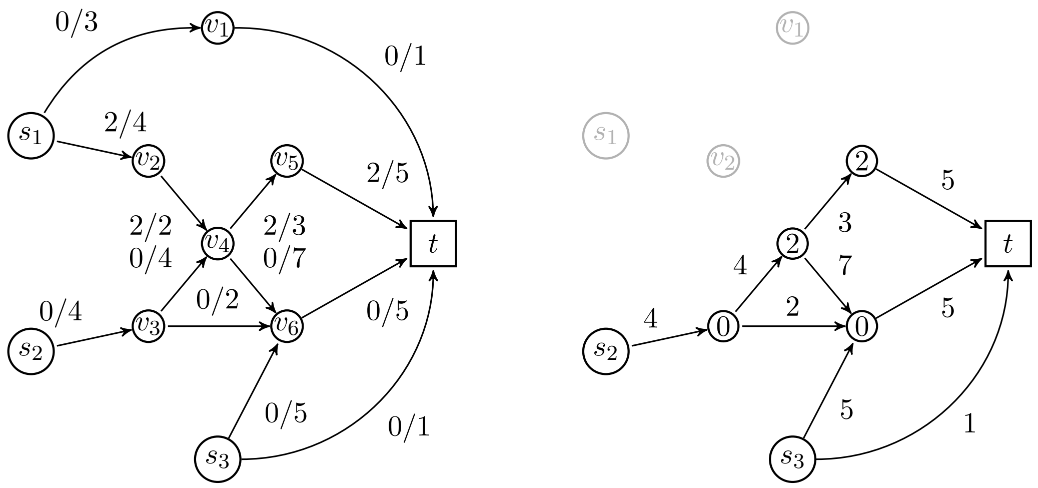

Figure 21: Nearly the same boundary instance as in Figure 19 with an additional arc from v_3 to v_6. On the right side, the corresponding residual network with node labels is depicted. We may route 3 flow units from s_2 via v_4, but this will pull 2 additional flow units to the arc (v_6, t), which attains its maximum capacity. Hence, v_3 may only send one flow unit directly to t. More flow can be reached, if we consider $\text{lmf}(s_2, 2, 0)$, i.e. $r = 0$. Now, we are not allowed to visit $v_4 = w_2$. Thus, we do not pick up any flow, the path with the arc (v_3, v_6) is chosen and the arc (v_6, t) is only loaded with two units of flow. Therefore, s_3 can send 3 additional flow units via v_6. However, choosing $\text{lmf}(s_2, 1, 2)$ and $\text{lmf}(s_3, 5, 0)$ will yield the optimal solution of 8 flow units for this choice of the first path. Still, one more flow unit could be reached, if we had considered the upper path for s_1.

Proof. First, we prove that the algorithm computes a feasible confluent flow and returns its flow value. For each source, the algorithm computes a single path which is a necessary condition. Thus, we have at most k paths. We have to show how these paths are merged correctly when they meet. We process the sources in strict cyclic order, thus it suffices to show that a new path is merged with the path computed in the previous recursion step. It cannot meet any other path computed so far without meeting the previous one, cf. Lemma 3.37. This is realized by the Update Procedure 3.42.

The new path cannot meet the previous path before its last bottleneck. With all outgoing arcs deleted this is a dead end. The path after the bottleneck arc is marked with the labels. If the path arrives at a node with a label $q(v) > 0$ for the first time, all further arc capacities are reduced by this value (cf. Definition 3.40). This conforms to re-routing this stored flow and finding a common bottleneck for the combined flow. Remember that labels are not additive. Furthermore, the previous path was computed by a left most flow. The update rule already deleted the left part of the network. Therefore, the new path will be right of the previous path.

Additionally, if we find an increased label, this implies that the previous path was merged with its predecessor path. Thus, the maximum label is propagated to route all

Algorithm 3.6: ConfluentBoundary

```
Input: residual instance $\mathcal{I}$
Output: maximum confluent flow value
1  if residual network contains no sources then
2      return 0;
3  Set $L :=$ sorted list of all residual capacities $(u(a) - q(v))$ $\forall a \in A$ $\forall v \in V$;
4  Determine $h$ such that $s_1 = w_h$;
5  Initialize $M$ as an empty list;
6  foreach $z \in L$ do
7      foreach $r \in \{0, \dots, h\}$ do
8          compute $(P, \bar{z}) := \mathrm{lmf}(s_1, z, r)$;
9          if there exists a left most flow for $z$ then
10             Compute residual instance $\bar{\mathcal{I}}$ (Procedure 3.42);
11             Rename sources of $\bar{\mathcal{I}}$ (decrease index by 1);
12             $z' :=$ ConfluentBoundary($\mathcal{I}$);
13             Insert $\bar{z} + z'$ in $M$;
14 return $\max M$;
```

stored flow in a confluent manner. Note that labels appear in ascending order due to the properties of the boundary instance. Finally, the update rule sets the labels to the current flow values along the path.

In summary, the computed paths never cross and are merged correctly. A confluent flow can be constructed by backtracking in the recursive Algorithm 3.6. Furthermore, the algorithm returns a value for the remaining instance plus the flow from the current source. Hence, also the flow value is calculated correctly.

Second, we show that our algorithm calculates a solution that is at least as good as an optimal solution. Given a boundary instance, let $\mathcal{T}$ be an in-tree of an optimal solution. First, compute a maximum confluent flow on $\mathcal{T}$ by the help of Algorithm 3.1. Choose the sources in ascending order, i.e. $s_1, \dots, s_k$. This does not change the value of the optimal solution, but it may change the flow distribution on the tree. Now, we compare this optimal solution source by source with the solution computed by Algorithm 3.6.

We start with s_1. Let s_i send z_i flow units in the optimal solution computed by Algorithm 3.1. We denote the unique paths in the in-tree $\mathcal{T}$ from s_i to t by P_i. Furthermore, this algorithm uses P_1 at its bottleneck capacity.

For $s = s_1$ and $z = z_1$ Algorithm 3.6 now computes $(P^1, \bar{z}) = \mathrm{lmf}(s_1, z_1, h)$ with $\bar{z}_1 \geq z_1$. Obviously, this path is left of P_1. Remember, that each path is left of itself. Hence, both paths can be identical.

Now, we continue with s_2. Consider P_2 in the residual instance and determine its bottleneck capacity b_2. Furthermore, let w be the node where P^1 and P_2 meet each other. Of course, we do not know P_2 in the algorithm, but since we also enumerate over r, there is a value r' with $w = w_{r'}$. We consider two cases:

1. If $\bar{z}_1 = z_1$, then $b_2 \geq z_2$ holds. Furthermore, our algorithm will compute $(P^2, \bar{z}_2) = \text{lmf}(s_2, b, r')$. By definition of the left most flow in a residual network, this path P^2 is left of P_2, since both parts of P^2 (before and after $w_{r'}$) are left of the corresponding parts of P_2. Additionally, $\bar{z}_2 \geq z_2$.

2. If $\bar{z}_1 > z_1$, then $b_2 < z_2$ is possible. However, our algorithms again computes $(P^2, \bar{z}_2) = \text{lmf}(s_2, b_2, r')$. Similarly, the computed path is left of P_2. Furthermore, any gap of $\bar{z}_2$ to z_2 is compensated by the overachievement of s_1 with $\bar{z}_1$ flow units. To be more precise, $z_1 + z_2 \leq \bar{z}_1 + \bar{z}_2$ holds.

For $k = 2$, this already yields the optimal solution. For the succeeding sources, we continue iteratively with the same procedure. For a suitable choice of w_r, we compute a path P^i which does not meet P^{i-1} before the path P_i from the optimal solution would meet it. Thus, among many sequences of values r, the algorithm also considers a certain sequence $(r_1, \ldots, r_k)$ and a certain sequence $(z_1, \ldots, z_k)$ such that P^i meets P^j in the same node where P_i would meet P^j $\forall j < i$. Furthermore, $\sum_{j=1}^{i} z_j \leq \sum_{j=1}^{i} \bar{z}_j$ holds. Since P_i provides enough capacity – it is a path in the optimal solution and P^i does not collect more flow on the boundary than P_i would do – it follows that P^i is left of P_i.

Thus, among many other solutions, our algorithm computes a sequence of paths P^i with bottleneck capacities $\bar{z}_i$ such that P^{l-1} is left of P^l and $\sum_{i=1}^{l} z_i \leq \sum_{i=1}^{l} \bar{z}_i$ $\forall l \in \mathbb{N}, l \leq k$. Whenever a source s_i does not send z_i units of flow like in the considered optimal solution, this missing flow was already compensated by the preceding sources. Therefore, for $l = k$, we have a solution that provides at least the optimal value. □

In easier words, Algorithm 3.6 enumerates over two parameters. The parameter r describes the topology of the in-forest of the confluent flow, i.e. it chooses the distance from the sink, where two neighboring paths P^i and P^{i+1} should meet. Simultaneously, the parameter z considers all reasonable flow values on these paths. Both parameters influence the actual in-forest that is computed. Among all optimal in-forests the one 'as left as possible' is chosen.

Theorem 3.45. *Let G be a planar graph with all terminals on the boundary and a single sink. If the number of sources is bounded by a constant k, then the* Maximum Confluent Flow *can be solved efficiently.*

Proof. We have already provided Algorithm 3.6 and its proof of correctness (see Lemma 3.44). Thus, we only need to discuss its running time. Obviously, the algorithm has a recursion depth of k, since one source is eliminated in every recursion step. At each function call, we have to compute up to $\mathcal{O}(kn^2)$ left most flows in the residual network. One left most flow computation can be done in polynomial time, e.g. by a simple depth-first-search strategy using the embedding. Assume, we have a counter-clockwise ordering of the incoming and outgoing arcs of each node, this can be calculated even in linear time. Furthermore, we use a sorted list of all arc capacities. This list has to be sorted only once and can be updated accordingly.

Hence, the running time of an algorithm following this approach contains at most a factor $(kn)^{2k}$, i.e. Maximum Confluent Flow can be solved in $\mathcal{O}(n^{2k})$. □

Note that the presented running time is a very rough estimation. Furthermore, Algorithm 3.6 can be accelerated in various parts. For example, the computation of left most flows in the residual network will fail very often for higher values of z and several values of r. Furthermore, we can skip all values of z in the loop, that are smaller than or equal to $\bar{z}$ computed in the last iteration.

3.5 Conclusion and open problems

In the previous chapter and in this one, we have studied various aspects of confluent flows. We presented some complexity results and tried to tackle the dual problem via confluent cuts. We have demonstrated an exact pseudo-polynomial time algorithm and an FPTAS for Maximum Confluent Flow on graphs with bounded tree-width which matched with our hardness results. Furthermore, we presented algorithms for outerplanar graphs and graphs with all sources on the boundary, both restricted to a single sink. However, some questions remain unresolved.

Future research should look into approximation algorithms (not schemes, as those cannot exist) for Maximum Confluent Flow on general graphs or improve the inapproximation bound from Theorem 2.12. Maybe one can even show that Maximum Confluent Flow cannot be approximated within a constant factor. Restricted to uniform capacities, this is unlikely because of the 13.29-approximation by Chen et al. [29] for the All-or-Nothing variant of Maximum Confluent Flow.

Furthermore, improving the cut definition is an attractive challenge. Of course, computing the minimum capacity of such a matching cut is at least as hard as computing the maximum value of a confluent flow. Nevertheless, a confluent cut at least for planar graphs with strict duality would be very interesting. The gap of Proposition 2.21 can certainly be reduced.

Finally, one may shorten or even close the gap between easy and hard confluent flow problems. One could start with a proof of Conjecture 3.35. It could also be worthwhile to study *intersection graphs* (cf. [24]). For paths, the intersection graph H of a graph G can be defined as follows: for each path in G add a vertex to H. Two nodes in H are connected if and only if the corresponding paths cross each other. Since such paths have to be avoided in confluent flows, it is conceivable that an analysis of the properties of intersection graphs may also provide deeper insight in confluent flow related problems.

4 Traffic Signal Coordination

A reduction of traffic jams will certainly improve the quality of life in the cities. However, simply expanding the infrastructure is often impossible due to space limitations. Besides strengthening public transport an increase of network performance seems to be a suitable way to cope with the boosting traffic. While the roads themselves do not allow much control of the traffic flow, signalized intersections provide a lot of traffic signal parameters ready to be optimized. Especially the coordination of numerous traffic signals at different intersections in a network seems worthwhile. The main concept of a so-called 'green wave', i.e. the coordination of traffic signals along an arterial road, was already introduced by Adolph [1] in 1925.

In this chapter we present a short survey on traffic signal optimization. We start with fixing the notation, introduce basic optimization concepts afterwards and build a bridge to traffic assignment. We close with a discussion of the computational complexity of offset optimization.

As a main intention of this chapter the reader should gain insight in the different concepts of traffic signal optimization. Each solution has advantages and disadvantages and is affected by the different approaches of traffic engineers or mathematicians. Finally, this should enable the reader to classify a new model for traffic signal optimization that will be presented in Chapter 5.

4.1 Introduction to traffic signals

4.1.1 A brief history of traffic signals

The traffic signal is much older than the automobile. On December 10, 1868, the World's first traffic signal was installed near the British Houses of Parliament in London at the intersection of George and Bridge Street. This traffic signal was invented by the railway engineer J. P. Knight and with its semaphore arms it also resembled railway signals. For night use red and green gas lamps were installed. The signal was manually operated by a policeman, who could turn the latern with a lever. Unfortunately, the gas powered light exploded on January 2, 1869.

Surprisingly, this first traffic signal was already equipped with the typical red and green light representing stop and go. Why these colors are used is not completely clear. Most likely they were derived from those in maritime navigation. Navigation lights on ships are nearly as old as seafaring itself. The oldest known law was enacted by Byzantine Emperor Leo III the Isaurian in 740. It ordered a white light on anchored ships[6]. During the centuries several rules and systems coexisted, but with the upcoming fast steam ships at the beginning of the 19th century a uniform system had to be created. In 1847, the leading seafaring nation Great Britain took the initiative and established a green light for the starboard (right) side, a red light for the port (left) side of a ship and a white masthead light, all to be shown between sunset and sunrise and at poor visibility. Why the attendant admirals and commanders decided this way is not told,

[6]This rule is still in effect.

but it gathered worldwide acceptance and changed only little since then [81]. With the rule *priority to right* this choice of lights has the following implication. A ship coming from the right side has the right of way, a ship coming from the left has to give way. At night, seeing a red light means seeing the left side of a ship. Hence, this ship is coming from the right side, i.e. one has to stop. Seeing a green light means seeing the right side of a ship, which is therefore coming from the left and one may go ahead.

The modern electric traffic signal was invented in 1912 by the policeman Lester Wire from Salt Lake City, Utah. The next red-green electric traffic light followed in 1914 in Cleveland, Ohio. While the early traffic signals were operated manually, the first automatic signal got a US patent in 1917. The first traffic signal with a third color to provide a warning for color changes was also invented by a police officer. This design by William Potts was first installed in Detroit, Michigan in 1920.

In Europe, traffic signals were first established in Paris, France and Hamburg, Germany in 1922. A famous traffic signal is the traffic signal tower at the Potsdamer Platz, Berlin, Germany, installed on October 21, 1924 (see Figure 22). It was a five sided tower with a cabin for a policeman on its top. On each side there were a clock and four traffic lights: red, blue (instead of amber), green and white. The white light allowed crossing for pedestrians. The traffic signal even increased congestion due to curious onlookers in the first weeks in operation.

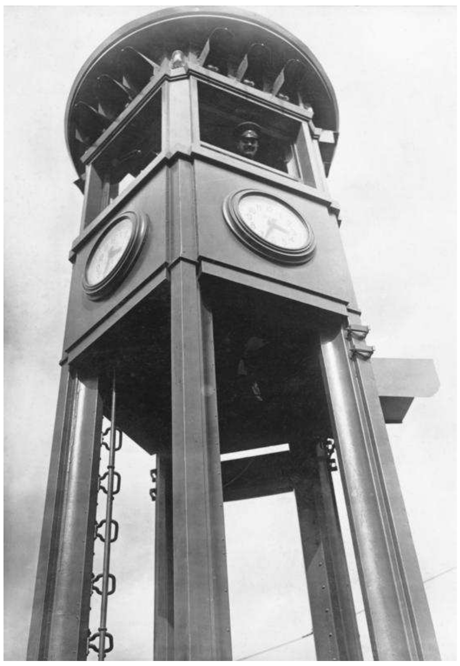

Figure 22: Traffic signal tower at Potsdamer Platz, Berlin, 1925. (Source: Bundesarchiv, Bild 102-01702)

Traffic signals in other metropolises followed, e.g. Milan in 1925, London and Vienna in 1926, Munich and Prague in 1927, Bremen and Nuremberg in 1928, Barcelona in

1929 and Moscow, Leningrad and Tokyo in 1930. Smaller cities like Helsinki or Basel obtained their first traffic signals in the 1950s.

In Europe the first traffic signals especially for pedestrians were established in Copenhagen, Denmark in 1933. But these traffic lights still resembled small sized traffic signals for automobiles. On February 5th, 1952 the first automatic pedestrian lights displaying the famous 'Walk' / 'Don't Walk' were set up in New York City. Meanwhile, Karl Peglau, Chief Psychologist of German Democratic Republic's Traffic Authority, thought of pedestrian lights suitable for children. On October 13th, 1961, he presented the first *Ampelmännchen* pictograms (see Figure 23). The traffic lights were upgraded with a mask that shows the silhouette of a man. The red man with his outstretched arms looks like a crossing guard, the green man is walking energetically. Starting from Berlin, the Ampelmännchen spread over the GDR. In 1982, it became a TV star in the regularly broadcasted children's bedtime television program *Unser Sandmännchen*. Once in a month, *Stiefelchen* and *Kompasskalle* entertained with stories about road safety. At the Czech festival for road safety education film in 1984, this series won the *Special Award by the Jury* and the *Main Prize for Overall Accomplishments* [77]. Four decades after its introduction, Daniel Meuren of the German weekly *Der Spiegel* described the Ampelmännchen as uniting 'beauty with efficiency, charm with utility, sociability with fulfilment of duties' [113]. Although the Ampelmännchen was the best pedestrian signal from a psychological point of view, it did not become a worldwide standard. Yet, optically poorer pictograms can be found in a lot of cities around the world.

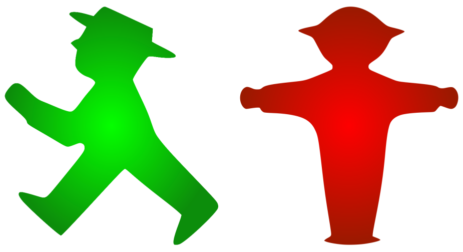

Figure 23: The *Ampelmännchen*.

4.1.2 The language of traffic signals

There is no unified terminology in the field of traffic engineering. Even when considering traffic signals there is no unified notation. John Q. Public is speaking of traffic lights, stoplights, traffic lamps, traffic signals, stop-and-go lights or even semaphores. But also the notation used by traffic scientists varies from country to country. Touching other scientific fields like mathematics, physics or psychology the number of terms used in traffic theory increases further. Fortunately, following this thesis only requires basic knowledge of traffic terms and we fix the notation in this section.

For further information, we refer to official handbooks, in particular the *Highway Capacity Manual* for the United States of America [146] and the *Handbuch für die Bemessung von Straßenverkehrsanlagen* (HBS) [60] as well as the *Richtlinien für Lichtsig-*

nalanlagen (RiLSA) [61] for Germany. There are also several good textbooks on traffic and traffic signals, e.g. [46, 69, 76, 130, 141]. More compactly, Warberg et al. [149] and Papageorgiou et al. [124] present surveys on traffic signal optimization.

Mathematically, we model the *road network* of a city as a directed graph $G = (V, A)$. The nodes $v \in V$ represent the intersections. The arcs $e \in A$ correspond to the connecting roads. Each arc e has a capacity $u(e)$ and a transit time $t(e)$ which is derived from its length and the speed limit.

Traffic signals are signaling devices positioned at road intersections (or other locations, e.g. pedestrian crossings) to control competing flows of traffic and to regulate the right of way.

Traffic signals at intersections are characterized by various parameters. The most important ones are *cycle time*, *red-green split*, order of the phases and the *offset* between traffic signals at adjacent intersections. Each single traffic light has a characteristic sequence of *red* and *green* that appears periodically with *cycle time* Γ. The proportion between red and green is called *red-green split.* All *traffic lights* at an intersection are grouped together to *signal groups* which again are grouped together to a *traffic signal.* The (interior) offset of their sequences is fixed to avoid collisions. These parameters are depicted in Figure 24.

Despite *amber* phases are depicted in the diagram, we do not use them further to avoid unnecessary complications. For our models we add this safety relevant times to the *red* phase.

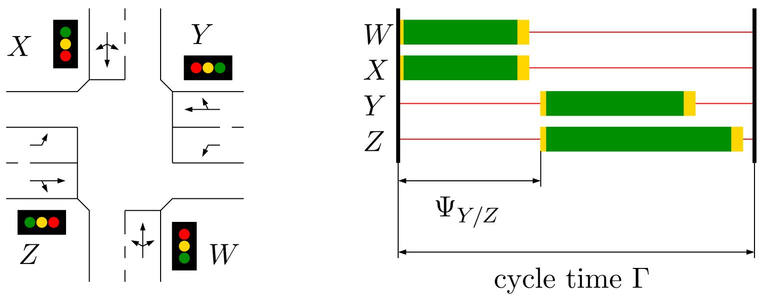

Figure 24: Signal plan for an intersection. Each of the four directions has its own signal group W, X, Y, and Z. Each group can consist of several traffic lights, e.g. a light for each lane. This signal has cycle time Γ. *Green* and *red* times are depicted in the diagram. At phase shifts there is also a short time of *amber.* The *green* phase of Z is extended, for example to enable left-turning. Time is measured relatively to a fixed point during one cycle. Ψ denotes the interior offset of a traffic light, i.e. the relative point in time this signal turns *green.* Thus, $\Psi_W = \Psi_X = 0$ and $\Psi_Y = \Psi_Z$.

If all traffic signals in a network have the same cycle time, one can also look at the *offsets* of the intersections with respect to a global system time. That means, given a fixed signal plan for an intersection, the offset of this intersection is the time span that the start of this plan is shifted with respect to the global system time. Let a car need t

units of time to bridge the distance between two consecutive intersections. If the *green* phase of the second traffic signal group starts exactly t time units after the *green* phase of the first one, the car experiences a *green wave*.

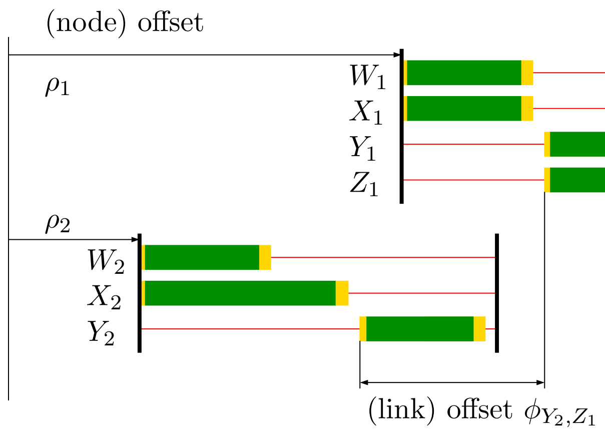

Figure 25: Signal plans for two traffic signals. The signals have offsets of ρ_1 and ρ_2 relative to a global system time. Therefore, the signal group W_2 at signal 2 turns green exactly $\rho_2 - \rho_1$ time units after the signal group W_1 at signal 1 does. Furthermore, it is assumed that one may travel from intersection 2 facing signal group Y_2 on link e to intersection 1 facing signal group Z_1. When the difference $\phi_{Y_2,Z_1} = \phi_e$ is near to the travel time *modulo* Γ on the corresponding link we have a progressive signal setting.

The offset parameters for our study are depicted in the *signal plans* in Figure 25. To determine all points in time a signal turns *green*, one has to add the (node) offset of the intersection, the (interior) offset of the signal group and an integer multiple of the cycle time. The *link offset* $\phi_{X,Y}$ denotes the relative difference between offsets of two consecutive traffic lights along a road (link).

All of these parameters can be used for traffic signal optimization. In this thesis we will concentrate on optimizing the offsets of all traffic signals in a network and we assume all other parameters to be fixed.

Additionally, we assume a unified cycle time at each traffic signal. Note that this is not a hard restriction; if traffic signals in the network have different cycle times, then the least common multiple of all cycle times can be used as a unified cycle time instead, which, on the other hand, may significantly increase the size of the model.

Let the street network be represented by a directed graph $G = (V, A)$ with node set V and arc set A. We will use the notation in Table 1. We also refer to the sequence of *red* and *green* lights as *operating sequence* of a traffic signal.

$n \in V = \{1, \ldots, N\}$	intersection indices
$e \in A = \{1, \ldots, M\}$	link indices
Γ	cycle time
ρ_n	(node) offset of intersection n
ϕ_e	edge offset on link e
$X, Y, Z, \ldots$	signal groups
Ψ_A^n	interior node offset of signal A at intersection n
$\Theta = \{\theta_1, \ldots, \theta_k\}$	a set of commodities with source node, sink node, and demand, i.e. $\theta = (s_\theta, z_\theta, d_\theta) \in V^2 \times \mathbb{R}^+$
$f_\theta(e)$	link flow for commodity $\theta \in \Theta$ on a link e
f_e	total link flow on link e, i.e. $f_e = \sum_{\theta \in \Theta} f_\theta(e)$
u_e	capacity of link e
t_e	free speed travel time on link e

Table 1: Notation for traffic lights and signals in a network model.

4.1.3 Safety constraints

The main intention of traffic signals is the safe guidance of road users at busy intersections. To avoid collisions at crossing streets several constraints have to be considered. Obviously, two crossing directions should never have *green* at the same time. Furthermore, there has to be enough time to clear the intersection at phase shifts (*clearance time*). The signal plan in Figure 24 takes this into account by a short time of *red* for all directions. In practice there are further simple rules for signal plans, e.g. each direction has to get *green* during one cycle. Other constraints are not that familiar, for example if a light switches to *green* there is also a minimum time before it may switch back to *red*.

Thinking of left-turners or pedestrians, one could write books about traffic signals at a single intersection. Actually, there are some books, e.g. [61, 146]. Furthermore, several rules are country-specific. Therefore, we leave the safety aspect to experts in this engineering aspect. With all interior parameters fixed by a traffic engineer, a safe crossing of an intersection can be guaranteed. Changing only offsets between consecutive signals is non-hazardous. In this thesis we will concentrate on this and mainly shift complete signal plans.

Additionally, this approach is also compatible to the infrastructure, commissioning, and traffic guiding principle of many cities. Due to their complexity, most intersections are designed independently. That is a few signal plans are developed for each intersection based on local traffic information. For safety reasons, the signal plans are encoded in read-only memory and the signal is operated by a micro-controller. Most signals are connected to a central server. Among other things, this server monitors the traffic signals and sends the global time signal. Therefore, it is easy to change (node) offsets. Changing other parameters will involve a complete, expensive analysis and evaluation of the affected intersection.

4.1.4 Controlling traffic signals

There are three major types of traffic signal systems: *pretimed*, *actuated* and *adaptive* signals. We briefly present their main properties and discuss their potential for optimization.

Pretimed signals. Pretimed signals use fixed signal plans. Most pretimed traffic signals have several plans for different daytimes. Assuming stable demand within a certain time, they provide an excellent chance of installing green waves. Furthermore, no additional hardware is needed. On the other hand, they cannot react to unexpected changes of traffic density.

Actuated signals. Actuated signals are based on pretimed signals, i.e. the phase sequence is also fixed in a signal plan. Additionally, they are equipped with detectors that measure the current traffic flow. They can lengthen or shorthen the *green* periods accordingly. So, they can respond to traffic changes. However, it becomes impossible to setup progressive signals. This can be partly repaired by fixing a common cycle time. That is for each lengthening one of the next phases has to be shortened to keep the cycle. In this case, a coordination in at least one direction is possible.

Adaptive signals. Adaptive signals completely rely on detector data. In the simplest setting, the traffic light turns *green* when a car arrives at the sensor. More intelligent systems also use historical detector data to make short time predictions of traffic volume and several signals communicate and work together. First and foremost, key aspects are reliable detection and stability of the system. Additionally, one hopes that these systems also provide low travel times for the road users. One main advantage of adaptive traffic signals is the possibility to integrate a priority for public transport. For instance, a phase can be skipped to directly give *green* to a bus or tram. Summarizing, adaptive traffic signals are very competitive at low traffic flow, e.g. at night. At rush-hour, the quality depends on the ability to cause green waves itself. This also means that an adaptive traffic signal setting falls back to a pretimed signal in this situation.

4.2 Traffic assignment

We cannot optimize traffic signals without discussing *traffic assignment*. Traffic assignment deals with route choices of the road users and describes the distribution of traffic in the road network. Traffic assignment can be seen from an administrative point of view, where a central authority tries to reduce the total travel time of the whole system. Or it can be seen from a game theoretical point of view, where each road user decides locally to optimize her/his experienced travel time. A precise definition will be given after the introduction of the necessary parameters. Obviously, there is a feedback between the problem to coordinate the traffic signals and assigning traffic units to paths in the network.

4.2.1 Motivation

Most road users are interested in the fastest route to their destination. Locals often know all the shortcuts in their home town. Assume, you have two routes to your destination of equal length when traveling with free speed. Obviously, your route choice will depend on the traffic volume on the particular route and the coordination of traffic signals along this route. If one of the routes provides a green wave, you will probably arrive much earlier. Consequently, every change of pretimed signal plans may influence the travel times in the road network and thus, it may also influence the road choice of the users.

The associated *aging* of pretimed traffic signal plans [15] is very important but often unappreciated in the optimization of traffic signals: It can be observed that some traffic signal coordinations, once having been very efficient, become worse over time. Increasing traffic in general and road users changing to optimized arterial roads lead to higher waiting times and may disturb the fine-tuned coordination.

Therefore traffic signal coordination and the traffic assignment should be considered as one single optimization problem. Hereby, traffic signal coordination means an optimal choice of the parameters of the traffic signals such that the road users reach their destinations as fast as possible. Traffic assignment describes the corresponding traffic pattern.

4.2.2 System optimal flows

We use *flow* and *multicommodity flow* to define traffic assignment. In the notation of Table 1, a flow is a function $f : A \to \mathbb{R}_{\geq 0}$ that has to fulfill capacity bounds and flow conservation constraints. We also require that the demands of all the commodities $\theta \in \Theta$ have to be satisfied. In the multicommodity case capacities are shared by the different commodities, i.e. $\sum_{\theta \in \Theta} f_\theta(e) = f(e) \leq u_e$. Furthermore, a travel time function t_e is given for each link.

A *traffic assignment problem* is the distribution of traffic flow in a street network satisfying demands of flow between origin and destination pairs of nodes. Assignment methods are looking for an optimal way to distribute the traffic flow in the network according to different objective functions. One option for this objective function is to minimize the total transit time spent by all flow particles in the network, $\sum_{e \in A} f(e)t(f(e))$.

In the following, we address the real-world application by using the notation *road user*, *traffic participant*, and *traffic flow*. These terms correspond to *flow unit*, *flow particle*, and *flow* in the theoretic model, respectively.

It matches to our experience that the travel time on a road is not independent of the traffic flow on this road. The more cars are on the road, the more time we need. Thus, $t_e(f(e)) : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is the travel time function describing the time a flow particle needs to traverse arc e in dependence of the amount of flow $f(e)$ on arc e.

In the literature, these travel time functions are also called *link performance function* or *latency function* [46, 69, 141]. Commonly, they are assumed to be non-linear, convexe, increasing functions (cf. Figure 26). Assume, we are equipped with link performance functions for each link. An assignment of flow to arcs, that minimizes the total travel time of all road users with respect to these travel time functions, is called *system optimum*.

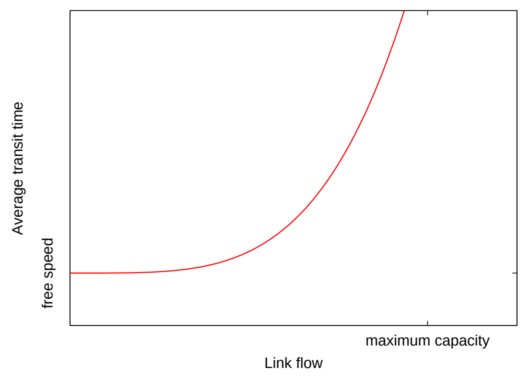

Figure 26: A typical link performance function. Starting at the free speed travel time, the average travel time increases with increasing flow. Sometimes the capacity bound on the link is implicitly forced by a pole at $f(e) = u_e$. Note that the link performance function does not capture traffic signal coordination or variation of flow values over time.

4.2.3 User equilibria and Wardrop's principles

While the system optimal solution is good for the system as a whole, it might be unfair to single users that could improve their own travel time by changing to a faster path. However, they will disturb traffic flow there and thus, the total travel time would increase.

In 1952, Wardrop [150] characterized the difference between a *system optimum* and an *user equilibrium* for a fixed origin-destination pair with two simple principles. His first principle reads as follows:

> *The journey times on all the routes actually used are equal, and less than those which would be experienced by a single vehicle on any unused route.*

A flow which fulfills this principle for each commodity is called *user equilibrium* flow. Please note that Wardrop considered uncapacitated networks. In contrast, *Wardrop's second principle* characterizes the system optimum.

> *The average journey time is a minimum.*

This concept of user equilibria is often used for modeling the behavior of traffic in street networks, where experienced users adjust their paths to minimize their own travel times until a stable situation is reached. A similar concept was developed by Nash at the same time [120]. Obviously, there is a gap between the value of the 'selfish' user equilibrium and the system optimum. This gap is sometimes called *price of anarchy*. A detailed survey on *selfish routing* can be found in [132].

A famous example for the impact of selfish routing is *Braess's Paradox* [23]. Braess constructed a small network with load dependent travel times. Surprisingly, adding a new link to this network increased the average travel time. Even more surprisingly, this effect was also observed in real world traffic, e.g. in Stuttgart, New York, and Seoul [99]. We present a version of Braess's Paradox in Figure 27.

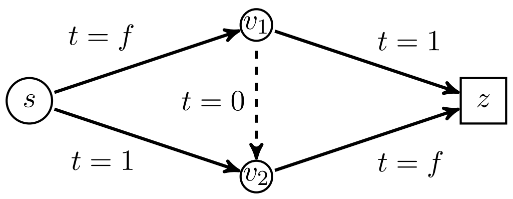

Figure 27: Example for Braess's Paradox. The additional road is dashed. Travel times are displayed at the links. Each road has capacity $u_e = 1$ and we send one flow unit from s to z, i.e. $\Theta = \{(s, z, 1)\}$. Without the new road we should send 0.5 flow units on the upper and on the lower road each. This yields a total travel time of $2 \cdot (0.5^2 + 1 \cdot 0.5) = 1.5$. With the new road, every flow particle can improve its own travel time from $f + 1$ to $2f$ by switching to the route (s, v_1, v_2, z). However, this will slow down the other particles on the links (s, v_1) and (v_2, z). Finally, all flow particles choose the new road, i.e. $f = 1$, yielding a total travel time of 2. Thus, the price of anarchy is $\frac{4}{3}$.

The problem of finding the user equilibrium is referred to as *equilibrium traffic assignment problem*. Due to the properties of the link performance functions (e.g. non-linearity), the traffic assignment problem cannot be solved by simply applying standard network flow algorithms but requires more involved approaches. For continuous, non-decreasing latency functions in uncapacitated networks, Roughgarden and Tardos [133] provide an existence theorem. Moreover, it can be shown, that a flow is at a *Nash equilibrium* if it solves the following convex optimization problem.

$$\begin{array}{lll}
\min & \displaystyle\sum_{e \in A} \int_0^{f(e)} t_e(x)dx & \\
\text{s.t.} & \displaystyle\sum_{e \in \delta^-(v)} f_\theta(e) - \sum_{e \in \delta^+(v)} f_\theta(e) = 0 & \forall \theta \in \Theta,\ \forall v \in V \backslash \{s_\theta, z_\theta\} \\
& \displaystyle\sum_{e \in \delta^+(s_\theta)} f_\theta(e) = d_\theta & \forall \theta \in \Theta \\
& \displaystyle\sum_{e \in \delta^-(z_\theta)} f_\theta(e) = d_\theta & \forall \theta \in \Theta \\
& f_\theta(e) \geq 0 & \forall e \in A
\end{array}$$

For non-decreasing link performance functions the objective is obviously convex. Roughgarden and Tardos [133] also provide bounds for the price of anarchy. Surprisingly, these bounds are independent of the network topology [131]. For affine linear travel time functions the gap between system optimum and user equilibrium is at most $\frac{4}{3}$.

Correa, Schulz, and Stier-Moses [37] extend the results of Roughgarden and Tardos to capacitated networks. Whereas flow values are often bounded implicitly by a pole in the travel time function in the former approach, we now consider a strict capacity bound. Consequently, Wardrop's first principle is no longer valid, since the capacity of the shortest paths may be reached first. Remaining flow particles have to choose longer paths. Even in the case of linear travel time functions, the value of *capacitated user equilibria* is no longer unique and the price of anarchy, i.e. the ratio of cost of the worst user equilibrium to that of the system optimum, jumps to infinity. Correa et al. characterize *capacitated user equilibria* by the non-existence of *unsaturated paths* that are shorter than any path actually used. They focus on a special convex sub-class of equilibria, based on a mathematical programming approach by Beckmann, McGuire and Winsten [14].

Please note that the described static traffic model does not capture any time-dependent behavior. For example, (static) link performance functions cannot handle platoons of traffic arriving at different points in time. All flow particles on a particular road are assumed to experience the same travel time. Thus, these static link performance functions are also not capable of capturing the dynamic properties of coordinated traffic signals.

4.2.4 Dynamic Flows

So far, the presented static assignment techniques conflict with the dynamics of traffic signals. Therefore, models are needed that can capture the development of flow in a timely fashion. One approach are *dynamic flows* or *flows over time*. Already Ford and Fulkerson [59] considered flows with a time component. An excellent survey on dynamic flows is presented by Skutella [142].

Now, a flow over time f with time horizon T is a (Lebesgue-integrable in the second parameter t) function $f : A \times [0, T) \to \mathbb{R}_0^+$. $f(e, t)$ is the *rate of flow* or *load* entering link e at time t. Flow particles entering e at time t arrive at the head of e at time $t + t_e$. The amount of flow that passes a link e can be calculated as $\int_0^T f(e, t)dt$.

We now have two possibilities to introduce a capacity for dynamic flows. We may simply restrict the flow rate, i.e. $f(e, t) \leq u(e)$. On the other hand, we may restrict the amount of flow that enters a link during a certain period of time, i.e. $\int_{\bar{t}}^{\bar{t}+\tau_e} f(e, t)dt$ $\forall \bar{t} \in [0, T - \tau_e)$. Thus, there can occur peaks in the flow rate which balance over time. Dynamic flows with such a capacity constraint are also called *bridge flows*.

Furthermore, *flow conservation* has to ensure that flow cannot leave a node before it arrives there. Additionally, one assumes that flow can be stored in a node for some time. In detail, for a non-terminal $v \in V$, $\sum_{e \in \delta^-(v)} \int_0^{\bar{t}-t_e} f(e, t)dt \geq \sum_{e \in \delta^+(v)} \int_0^{\bar{t}} f(e, t)dt$ $\forall \bar{t} \in [0, T)$. If we require a strict equation, then flow must not be stored at intermediate nodes. Demands are handled similarly.

If the flow is induced by a static flow, i.e. it uses only s-z-paths $P_{s,z}$ with a constant flow rate in the time interval $[0, T - t(P_{s,z}))$, then the dynamic flow is called *temporally repeated flow*.

In this setting, various interesting problems appear. In the MAXIMUM FLOW OVER TIME problem, we want to send as much flow as possible from a source s to a sink z until the time horizon T is reached. This problem can be solved with the Ford-Fulkerson algorithm for static maximum flows. In detail, one computes a path decomposition of a maximum flow on a slightly modified network and derives a temporally repeated flow.

Earliest arrival flows are a variant of this problem. The goal is to find a single s-z-flow over time that simultaneously maximizes the amount of flow reaching the sink z up to any time $t \geq 0$. Surprisingly, such flows exist. However, it seems unlikely that they can be computed in polynomial time [142].

The QUICKEST FLOW problem corresponds to a transshipment in minimum time. For a single source and a single sink, it can be computed efficiently by maximum flows over time and a Megiddo search of the time horizon [26]. Surprisingly, for multiple sources or sinks, this problem is not equivalent to a maximum flow over time. However, Hoppe and Tardos present an efficient algorithm [78, 79].

Minimum cost flows over time are $\mathcal{NP}$-hard [90]. Considering multiple commodities, MAXIMUM FLOW OVER TIME is also hard [75]. Klinz and Woeginger [91] study a slightly modified integral version of dynamic flows. Here, arcs are called *dedicated* if flow blocks an arc as long as the transmission continues. This also yields $\mathcal{NP}$-hard problems.

Many dynamic flow problems can be solved using *time-expanded networks*. Already Ford and Fulkerson introduced time-expanded networks in their seminal work on network flow theory [59]. In this expanded network, for every node, several copies of this node are added to the graph, one for each desired time step. These nodes are connected by arcs, where the various copies of the vertices are connected accordingly to the travel times of the original arcs. Additionally, arcs connecting consecutive copies of the same node may allow waiting. We give a precise definition and Figure 32 shows a simple network together with its time-expanded network.

Definition 4.1 (Time-expanded network). *Let $G = (V, A)$ be a network with capacities $u : A \to \mathbb{N}$ and non-negative integral transit times t_l. For a given time horizon $T \in \mathbb{N}$, the corresponding time-expanded network $G^T = (V^T, A^T)$ is constructed as follows.*

1. *For each node $v \in V$, we create T copies $v_0, v_1, \ldots, v_{T-1}$, thus $V^T = \{v_t | v \in V, t \in \{0, 1, \ldots, T-1\}\}$.*

2. *For each arc $e = (v, w) \in A$, we create $T - t_e$ copies $l_0, e_1, \ldots, e_{T-t_e-1}$ where arc e_t connects node v_t to node w_{t+t_e}. Arc e_t has capacity $u(e_t) = u(e)$.*

3. *Additionally, there are waiting arcs from v_t to v_{t+1} $\forall v \in V$ and $\forall t \in \{0, 1, \ldots, T-2\}$.*

The big advantage of flows over time compared to static flows is the opportunity to capture the complete time-dependent behavior of the journey of a flow unit in a given network. In contrast, a static flow can only describe the time-independent behavior of this journey. This yields a very powerful approach and it was already suggested to use flows over time in traffic networks [94] or logistics [57]. Our new model for traffic signal

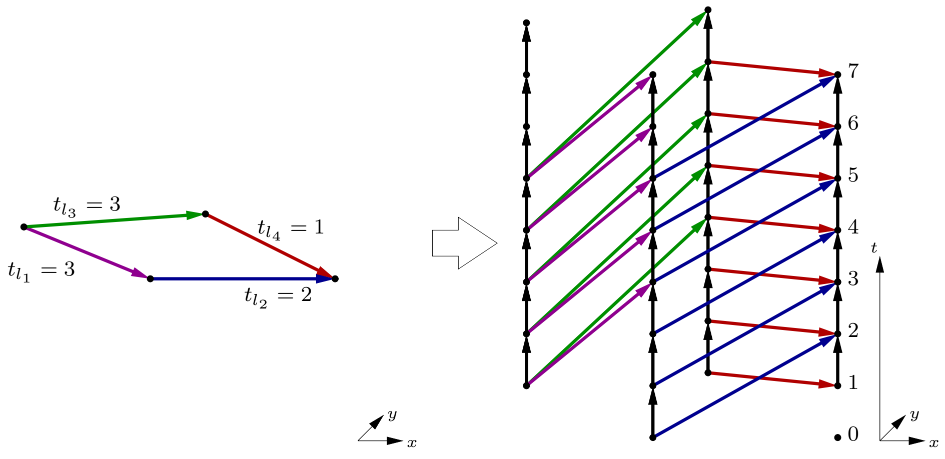

Figure 28: Simple example of a 4-vertex graph together with its time-expanded network for time horizon $T = 8$. The network is presented in a perspective view. The graph itself is embedded in the xy-plane. The time is displayed along the vertical t-axis. Corresponding arcs have the same color. Waiting arcs (black) allow buffering of flow units in a node for some time.

coordination in Chapter 5 will be based on this concept. Unfortunately, the model for dynamic flows quickly becomes very large and a different input (e.g. dynamic demand) is needed. Thus, we cannot apply this approach one-to-one to traffic signals.

4.2.5 Dynamic traffic assignment

Dynamic traffic assignment model (DTA; see, e.g., [34, 144] for surveys) is a general term for several similar approaches which try to extend traffic assignment with static link performance functions to a dynamic model with a time component. The travel time which a road user experiences on a road now depends on the arrival time of this traffic participant at the road and the actual traffic flow on this road at this time. Consequently, a road user chooses its route and a departure time.

DTA models have in common that they try to compute a user equilibrium with an iterative approach. However, the methodologies are quite different, but they share a similar overall model structure. One may identify three major steps [34]. Firstly, given a set of route choices, the resulting travel times are computed. Various network loading models including both analytical and simulation-based approaches are used in this step. For example, several queueing models are tested [144]. Secondly, the new shortest routes are computed for each origin-destination pair and each departure interval with respect to the new link travel times. Thirdly, given the updated route sets, vehicles are assigned to new routes. If the stopping criterion is not met, then one returns to the first step.

Summarizing, these models focus on varying demand over time, but they still use standard travel time functions. Unfortunately, as pointed out in [144], the queueing models lead to non-monotone and non-differentiable route travel times. Hence, solutions, if they exist at all, are hard to find, non-unique, and heuristic solution methods are suggested. Furthermore, some DTA models cannot ensure that a user on a link who enters the link earlier than another user will also leave it sooner than the second user.

Additionally, these approaches do not–to the best of our knowledge–capture changing coordinations of traffic signals. Some of them do not provide the accuracy to handle the very fast changes in traffic density caused by traffic signals. But traffic signals and their coordination are the main reason for platoons of cars in inner-city traffic. Thus, there is a time-dependent change of flow on an even finer level. It seems to be essential that a model for traffic signal coordination also captures these platoons.

4.2.6 Computing assignments

As already mentioned in the introduction (Section 1.2.2), several combinatorial algorithms are known for the single commodity maximum flow problem. Similar approaches can be used to solve assignments with linear cost functions, but additional difficulties occur. In the style of Ford and Fulkerson, one may use a residual network and augmenting paths. Here, cycles with negative total cost are a crucial problem. These cycles can be canceled by pushing flow backwards on them. Always augmenting flow on cycles with minimum mean cost yields a polynomial running time. Alternatively, negative cost cycles can be avoided by using the successive shortest paths algorithm which yields a pseudo-polynomial running time. Using capacity scaling, this algorithm can be used to compute minimum cost flows efficiently. A detailed presentation of the algorithms mentioned in this section can be found in [2].

Minimum cost flows are easily formulated as a linear program and thus, they can be computed with the simplex method. This approach can be accelerated significantly by exploiting the special structure of the underlying network problem [2]. Given a flow x, an arc e is called *restricted arc* if $x(e) = 0$ or $x(e) = u(e)$. A flow x is a spanning tree solution if there exists a spanning tree such that every non-tree arc is a restricted arc. One can show that the problem always has an optimal spanning tree solution. The *network simplex algorithm* (see, e.g., [2]) starts with a feasible spanning tree solution and improves it iteratively maintaining this property until it becomes optimal. Besides linear programming multicommodity instances can be solved with approximation algorithms, e.g. [67].

Convex cost functions can be approximated with piecewise linear functions. Subsequently, an arc with a piecewise linear cost function can be transformed to a set of parallel arcs with different linear cost functions. For each linear piece of the original function, an arc with a linear cost function with the same slope is used. The capacity is adjusted to the length of the interval of the linear segment. If a small amount of flow is assigned to these arcs, only the cheapest arc, i.e. the one with the lowest slope, is used. If the flow value is increased, then the arcs with higher cost functions have to be used, too. Combining both steps with an algorithm of the first passage of this section yields a combinatorial approximation algorithm for the minimum cost flow problem with convex cost functions. Due to the substantially increased network size, the running time of this algorithm is not polynomial in the input size. A generalization of the capacity scaling algorithm with successive shortest paths solves the convex flow problem efficiently [2]. Alternatively, convex programming can be used to handle convex cost functions exactly [122].

Besides these combinatorial and mathematical programming approaches, there exist various heuristic methods using, e.g., simulation and genetic programming for determining traffic assignment solutions. The agent-based simulation tool MATSim (cf. Section 4.3.5) is an example for one of these strategies, where assignment is realized by an iterative simulation and replanning. On the one hand, the traffic assignment approach in MATSim is similar to some DTA models. On the other hand, MATSim provides features like public transport, traffic signals and many more. Thus, although we only summarize the assignment approach here, MATSim should not be reduced to a pure assignment model.

Actually, the non-cooperative road users (also known as *agents*) of MATSim play a game on the underlying network to find user equilibria. In the simplest case, each link has a constant travel time and it is modeled by a queue. An agent is directly assigned to this queue when it enters the link and it has to stay in the queue until the travel time has expired. Further, the storage of a queue and the number of agents that can leave a queue in a certain time interval are limited. The queue also guarantees the *first in first out* property. Hence, agents can move through the network in a timely fashion. The model is extended with lanes, intersections and traffic signals.

Each agent is equipped with a set of *plans*. Each plan consists of activities, i.e. a destination in the network and a duration, and a route connecting these activities. In every iteration step, each agent chooses one plan with respect to a score for execution. All chosen plans are evaluated simultaneously in a simulation run. The experienced travel time is used to update the scores of the plans. Afterwards, a fixed amount of road users may change their plans by varying the departure time or computing an alternative route. This process is repeated until a convergence criterion is reached.

Summarizing, MATSim is built-up around an evolutionary algorithm and the assignments computed by MATSim can be interpreted as stochastic user equilibria [117].

This concept admits to compute user equilibria for very complex systems where not all relations are known in terms of a closed mathematical formulation. However, discussing these approaches more detailed we would slightly drift away from a strict mathematical modeling of traffic flow and simulation issues would come into play.

4.3 Optimizing traffic signals

The idea of coordinating traffic signals is nearly as old as the traffic signal itself. As mentioned in the introduction of this chapter, Adolph [1] patented his concept of ‘green waves’ in 1925. But it was not before 1964, when Morgan and Little [116] started a broad analysis and presented a graphical solution for calculating maximal time slots and bandwidths for a single road.

By now there exist various models to transform the concept of a ‘green wave’ to a whole network of roads. It is impossible to discuss all of them here. However, we will give a survey on the–to the best of our knowledge–most important and pioneering approaches.

In the last decades, a number of models were presented, allowing for different phenomena of urban traffic. The concepts in this section can be distinguished by various points of view. We will present heuristic approaches as well as strict mathematical programming approaches. Other models use direct search techniques. The models will address different parameters (offset, red-green split, cycle length and combinations thereof), scenarios (single intersections, arterial roads, networks), and signals (pretimed, actuated, adaptive). They try to optimize different objectives (minimizing travel time, delay, number of stops, fuel consumption, and combinations thereof; maximizing greenbands). Furthermore, we have to distinguish between theoretical and practical approaches.

4.3.1 Measures of performance

The performance of traffic signal optimization can be measured by different parameters. One of the first measures used in the literature is the *bandwidth* of *greenbands* (cf. Figure 29). A greenband consists of several consecutive intersections on a certain route and a time interval such that all road users arriving at the first signal during this time slot experience a green wave on this route. Bandwidth is the width of the greenband, i.e. the time span in which the green wave is available. However, it is difficult to apply this concept on networks. Nowadays, the most common measure is the *average delay* or the *number of stoppages*. A weighted combination of those has also been used.

When considering the traffic assignment problem and an optimal traffic signal coordination simultaneously, however, measuring delays and stoppages is not a good choice. Traffic participants may now choose arbitrary routes. Thus, taking only stops and delay into account, a road user may choose a very long detour through the network just to avoid stopping or waiting in front of a *red* traffic light. This is rather unrealistic as most road users are interested in the fastest way to their destination. Therefore, one may choose the total or average travel time (transit time + delay) as the measure of quality of the solution. However, this measure is rarely used in the literature.

Furthermore, Sun et al. [143] have shown that the number of stops and delays are very sensitive to changes of the cycle time. Long cycle times minimize stops while short cycle times lead to less delay. Thus, if combining these two measures in an objective function, slight changes in the weighting may lead to very different coordinations.

There are also several other measures of performance, e.g. preferences for public transport and pedestrians or fuel consumption, that are not discussed in this thesis.

4.3.2 Optimization of pretimed signals

Assuming fairly stable demands within certain divisions of time, the usage of pretimed traffic signals appears reasonable. All approaches presented in this section introduce a simplified traffic model and suggest a solution method. However, they vary significantly in the model assumptions and the optimization techniques.

In 1964, Morgan and Little [116] presented MAXBAND, a graphical method for maximizing *bandwidth*. The concept of bandwidth is displayed in Figure 29.

Little [106] developed this approach further, using mixed-integer programming. This

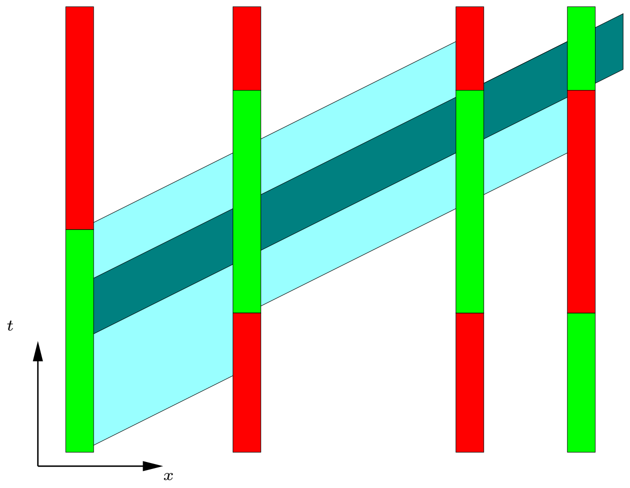

Figure 29: Time-space-diagram showing the coordination of four consecutive traffic lights. Many traffic planning tools visualize signal coordination in this way. The distance is displayed on the horizontal axis, the time moves on in vertical direction. Cars in the light blue area arrived at all predecessing signals at *green*. But only in the dark blue zone, there is a green wave through all four signals. This area is the *greenband*. Its width is called *bandwidth*. Some optimization approaches like MAXBAND try to maximize this parameter.

program adjusts optimal values for offsets and a common cycle time for a bi-directional road. Two years later, a graph-theoretical model for minimizing delay was developed by Allsop [4]. Hereby, the solution is successively extended from a solution of a sub-network.

Shortly after, Robertson [128] presented his theoretical work on the optimization of offsets. He uses a simplified simulation approach and a genetic algorithm to optimize this traffic signal parameter. These results led to the development of TRANSYT[7], until now a most widely used tool for inner-city traffic optimization. TRANSYT provides mainly two different heuristic techniques to improve a so-called performance index. It is presented more detailed in Section 4.3.5.

Gartner et al. [68] presented a mixed integer programming approach for network coordination in the mid-1970s. They developed MITROP[8], which was one of the first approaches for networks that used integer programming. Gartner et al. introduce saturation deterrence functions to capture the influence of offsets. Further, they propose a piecewise linear approximation of these functions such that a mixed integer linear program can be formulated. The integer programming formulation also contains cycle constraints for each cycle in the network, i.e. the algebraic sum of the edge offsets along this cycle has to be an integral multiple of the cycle time. The importance of these equations is easy to understand. Assume to go for a round trip on an arbitrary cycle. Let ϕ_i be the ith edge offset on this cycle, i.e. the second signal turns to *green* exactly

[7]TRAffic Network StudY Tool
[8]Mixed Integer TRaffic OPtimization

ϕ_1 time units after the first signal. The third traffic signal turns to *green* ϕ_2 time units after the second signal and thus, $\phi_1 + \phi_2$ time units after the first signal. On the one hand, if the round trip is finished and we are standing in front of the first traffic signal, then all edge offsets have summed up. On the other hand, the first traffic signal is still in cycle with itself, i.e. if it turns *green*, then a certain number periods have passed by. Hence, cycle constraints ensure the consistency of edge offsets. As a consequence, an integral variable is needed for every cycle in the network.

In 1977, Allsop and Charlesworth [5] presented an example that coordination and assignment interact. They suggest an iterative approach for optimizing both problems.

Serafini and Ukovich [139] studied the scheduling of periodic events. As an outcome, they also presented an involved mathematical model for the coordination of pretimed traffic signals [140] in 1989. In their context, phase changes of traffic signals are events. The authors define several kinds of constraints for signals, conflicts, coordination, and cycles. More precisely, they introduce a lower and an upper bound for the link offset on each road. Thus, their approach is a feasibility problem.

Recently, Wünsch et al. [95, 115] presented a further development of the approach of Gartner et al. [68]. They replace the saturation deterrence functions by a more realistic platoon model. In their model flow units are bundled during the *red* phase of a traffic light. When the signal switches to *green*, the flow particles resume their journey together as a platoon. Hence, they will also arrive at the next intersection as a platoon. This approach admits a very realistic estimation of waiting times at traffic signals which are derived as the integral over the queue length. For a fixed assignment, i.e. the waiting time only depends on the offsets, a piecewise linear approximation of these offset induced delays is used to turn the problem formulation into a mixed integer program. Here, Wünsch [154] also focusses on the cycle constraints introduced by Gartner et al. [68]. Cycle constraints apply to every cycle in a network and street networks constists of a huge number of cycles. Wünsch proves that it is sufficient to demand cycle constraints on a subset of cycles which span all other cycles in the network. To accelerate the mixed integer programming, he searches for minimal cycle bases which reduce the number of integer variables.

4.3.3 Adaptive traffic signals

It is often seen as a handicap that pretimed traffic signals cannot react to fluctuations in traffic flow, whereas traffic itself is seen as a stochastic process. Consequently, in the last decades, big efforts have been made on developing highly adaptive traffic signals which are able to react on changes in traffic volume immediately. The approaches differ significantly from those of pretimed signals.

On the one hand, adaptive signal control can be seen as a problem of interdisciplinary *control theory*. The usual objective of control theory is stability of the dynamical system. A controller reacts on the input and tries to manipulate the system such that it follows a *reference* without oscillating. The same is true for adaptive signal control. Primarily, the system tries to avoid building up congestion. Furthermore, even a single road user should get *green* within a certain amount of time. Optimization is more or less a secondary goal. Furthermore, real-time feasibility is a critical aspect for the applicability of these techniques in practice.

On the other hand, one can also account the problem to *online optimization*. Thus, all adaptive control strategies are heuristic techniques based on limited information. However, hardly any analysis concerning this point of view can be found in the literature. It would be very interesting to compare the heuristic solutions to optimal offline solutions.

Most solutions consist of a traffic model and a complex system of detectors and traffic signals. They focus on the following questions:

- Which detectors are used?
- Where are these detectors placed?
- Which traffic signal gets which information?
- How does the signal react on the information?

These questions also include that information on traffic is propagated in the traffic network in space and in time. Some systems also use historical data to predict future traffic. Adaptive signal control techniques are often evaluated with the help of micro-simulations which yields very realistic results.

There are several competing approaches for optimizing adaptive traffic signals. A survey of common approaches is, e.g., presented by Friedrich [63, 64]. Representatively, we discuss two recently published systems.

The first model is based on a cell transmission model for networks developed by Daganzo in a series of papers [39, 40, 41, 42, 43]. The underlying cellular automaton model was qualified for traffic theory by Nagel and Schreckenberg. In their famous paper [119], the authors explain spontaneous congestions on freeways with help of the modified one-dimensional binary cellular automaton *rule 184*. The name of the rule is a *Wolfram code*. It defines the evolution of the states of the cellular automaton (cf. [105]). Daganzo extends this approach to traffic networks with intersections. Further, he studies gridlocks in this system. Almasri and Friedrich [6] extend this model with an adaptive signal control. A genetic algorithm is used for optimization. Due to the high computation time the limit for practical applicability is already reached for the comparatively small test network with six intersections.

The second approach was recently presented by Lämmer [102]. The three basic components are a local prediction of future traffic, a local optimization with a priority index for the next phase, and a local stabilization to force an average and a maximal period. This system is going to be tested in reality at seven intersections in Dresden, Germany. Lämmer states stability as one of the main problems of adaptive systems. He suggests an underlying pretimed signal coordination as a guideline to stabilize the system. Further, he states that an adaptive system has to be runned below saturated traffic demand to prevent degeneracy [103]. Therefore, offset optimization is also important for the operation of adaptive traffic signals

4.3.4 Simultaneous coordination and assignment

Most of the approaches presented so far did not consider assignment. But each change of a traffic signal parameter will also change travel times for at least a few road users. Allsop and Charlesworth recognized the feedback between an optimized coordination and traffic assignment [5]. In 1977, they proposed an iterative approach. Signal timings are optimized with TRANSYT. Afterwards, an equilibrium traffic assignment is computed. These steps are repeated until no change in the coordination occurs. Obviously, using heuristic methods iteratively, one may only hope to derive a local optimum. Even worse, this approach may lead to a decline of network performance as shown by Dickson [48].

Despite the interaction of signal coordination and traffic assignment, only few research was spent in the combined optimization. For example, Chiou [31, 32] presented a bilevel formulation based on the approach of Allsop and Charlesworth. She uses a gradient projecting method for calculating local optima. Further, she presents a subgradient method for a non-smooth single-level formulation of the problem [33]. Recent results were also made by Bell and Ceylan [16] as well as Teklu et al. [145] using genetic programming.

4.3.5 Tools

So far, we have seen several (theoretical) approaches for traffic signal optimization. In this section, we will present some tools and software kits, that are actually used by traffic engineers to optimize and control traffic signal systems.

Optimization. TRANSYT is a widely accepted software tool for modeling, analyzing, and optimizing traffic signal settings. It is based on an approach by Robertson [128]. It is developed by the Transport Research Laboratory. TRANSYT supports several sophisticated traffic models, including the Cell Transmission Model and the Platoon Dispersal Model, to derive a so-called *performance index* (PI). Normally, this index is a parametrized sum over stoppages and delays in the network. TRANSYT uses optionally a genetic programming approach or a gradient search technique for improving this performance index. On the one hand, the detailed objective function is seen as a big advantage of TRANSYT since also other aspects like fuel consumption can be taken into account. On the other hand, the obtained solution significantly depends on the parameter choice of the user.

Another widely used traffic planning tool is VISUM, developed by ptv AG in Karlsruhe, Germany. It provides several tools for all kinds of inner-city traffic from a single intersection up to a whole town. VISUM features various state-of-the-art static traffic assignment techniques. The tool also supports the optimization of traffic signals, e.g. by a plug-in based on the approach of Wünsch. Further, VISUM provides tools for planning public transport or for estimating pollution.

Signal control. Of course, an offline optimization of traffic signals is not sufficient, the signal have to be operated in practice. We briefly discuss two tools that are actually used to control traffic signals. Both of them fall into the category of actuated signals.

Since both systems are based on pretimed signals this also emphasizes the need for well coordinated signals.

SCOOT[9] was primarily developed by Bretherton et al. [80] and presented in 1981. It uses TRANSYT to optimize green splits, offsets and cycle length in real time. Based on prevailing traffic conditions SCOOT modifies those parameters in small steps to keep track with changing traffic conditions. SCOOT is used all over the world and is still enhanced.

A similar system mainly used in Germany is Sylvia+ [135]. This method is also based on a coordinated pretimed signal coordination. Depending on the recent traffic, Sylvia+ can extend or shorten a single phase of the traffic signal. However, changing phase durations in an arbitrary manner will completely disturb the underlying coordination. Therefore, Sylvia+ uses a recovery algorithm to restore the common cycle. Whenever a phase is extended, another phase in the same period has to be shortened by the same amount of time. Thus, at least for one direction, a green wave can be maintained.

Simulation. Micro-simulation is essential in the development of new traffic signal coordination methods. Not every coordination can be tested in practice. We briefly present two simulation tools, namely MATSim and VISSIM, that we will also use to evaluate our approach for signal coordination in Chapter 6.

The software MATSim is mainly developed at TU Berlin and ETH Zurich and is a multi-agent transport simulation tool based on queue models. MATSim is a multi-purpose simulation tool. Among others, it can be used for a traffic microsimulation in inner-cities. It provides lanes, queues and traffic signals, but the dynamics of agents are reduced to their bare essentials. It is capable of simulating large scale traffic networks and computing traffic assignment using an iterative approach. See [112] for more details on this software.

The tool VISSIM from ptv AG provides a microscopic traffic simulation with state-of-the-art longitudinal dynamics and lane change models [126]. VISSIM is widely accepted to produce very realistic results. It also supports several additional features like pedestrians and public transport which are far beyond our purposes. VISUM and TRANSYT provide a link to VISSIM.

Despite their different simulation approaches, both tools provide measured travel times for the commodities and they visualize congestion. Thus, we can use at least these parameters to evaluate signal coordinations and to compare both tools.

4.3.6 Conclusions

There are many reasons for the usage of coordinated pretimed traffic signals. Considering rush hour situations with high demand at every road, also adaptive coordinations will fall back to fixed time coordinations. Furthermore, pretimed signal settings are the basis for widely used actuated signal systems. Additionally, due to the high costs of sensor technologies and a lacking acceptance of car-to-car or car-to-traffic-signal communication

[9]Split Cycle and Offset Optimization Technique

for data privacy reasons, a complete adaptive control is out of range for many cities. Thus, the importance of well coordinated pretimed signals will persist the next years.

On the one hand, most approaches for traffic signal coordination use heuristic methods to improve the signal settings. This is especially true for the combined optimization of signal coordination and traffic assignment. These models are somewhat unsatisfying from a mathematician's point of view. Despite the solutions operate well in practice, one has no guarantee of optimality. On the other hand, models designed for exact solution methods often make inaccurate assumptions on the traffic flow model. Thus, one can perhaps solve the model exactly, but the applicability is sometimes disputable.

4.4 Complexity of signal coordination

Closing the short survey on traffic signal optimization, we discuss the complexity of the *signal coordination problem* in this section. The formulation and analysis essentially depends on the chosen model. Various proofs of complexity for different approaches can be found in the literature. All of them have in common that the offset optimization problem for traffic signal coordination is *$\mathcal{NP}$-hard.* Network coordination is closely related to the Periodic Event Scheduling Problem (PESP). Serafini and Ukovich [139] proved $\mathcal{NP}$-completeness for the PESP by reducing the Hamiltonian Cycle Problem. Wünsch [154] provides a reduction from PESP to his formulation of the Network Signal Coordination problem.

In the following we show that traffic signal coordination is an $\mathcal{NP}$-hard problem also in our traffic model. This result is presented before the introduction of our model, since it also applies to more general models and it is not restricted to our approach. More precisely, we consider traffic signal coordination in a general dynamic network flows setting with constant transit times. The network contains traffic signals where waiting is possible. Building up on this assumption we define the signal coordination problem as follows.

Problem 4.2 (Signal Coordination Problem).
INPUT: *Network $G = (V, A)$ with arc capacities $u(e)$ and travel times consisting of a constant transit time $t(e)$ per arc plus waiting time at the intersection; a set of traffic signals with arbitrary but fixed operating sequences at some of the intersections $v_n \in V$, $n \in \{1, \ldots, N\}$; a cycle time Γ that is the same for all traffic signals; commodities $\theta \in \Theta$, $\theta = (s_\theta, z_\theta, d_\theta)$ with origin s_θ, destination z_θ, demand d_θ and fixed routes.*
OUTPUT: *A dynamic flow on the fixed routes and a set of offsets $\{\rho_1, \ldots, \rho_N\}$ which minimize the total travel time, i.e. the sum over the travel times of all road users.*

Theorem 4.3. *Offset optimization of traffic signals is $\mathcal{NP}$-hard, even with travel times consisting of constant transit time per arc plus waiting time at the intersection.*

Proof. First, let us fix the decision version of the SIGNAL COORDINATION PROBLEM: Given a network $G = (V, A)$ with capacities, travel times, commodities and traffic signals as in Definition 4.2 and a value t^*, is there a coordination (set of offsets) that induces a total travel time equal to or less than t^*.

We reduce the Signal Coordination Problem from the 3-Partition Problem which is $\mathcal{NP}$-complete [66]. The problem is to decide whether $3m$ integers can be partitioned into m sets of 3 integers of equal sum. More formally:

> 3-Partition Problem:
> Input: multiset $S = \{x_1, \dots, x_{3m}\}$ of $3m$ integers
> Question: Is there a partition of S into disjoint triples $S_1, \dots, S_m$ that all have the same sum $C = \frac{1}{m}\sum_{i=1}^{3m} x_i$?

The 3-Partition Problem remains $\mathcal{NP}$-complete when $\frac{C}{4} < x_i < \frac{C}{2} \ \ \forall i \in \{1, \dots, 3m\}$.

Given an arbitrary instance $S = \{x_1, \dots, x_{3m}\}$ of 3-Partition we show how this instance is used to build an instance of the the Signal Coordination Problem such that the Signal Coordination instance has a solution if and only if the 3-Partition instance has a solution.

Figure 30 shows the network used for the reduction. We create $3m$ roads with a demand of x_i, $(i = 1, \dots, 3m)$ flow units in each road $(C = \frac{1}{m}\sum_{i=1}^{3m} x_i$ and $\frac{C}{4} < x_i < \frac{C}{2} \ \ \forall i \in \{1, \dots, 3m\}$ as above). There is only one path to the destination for each commodity, thus route choice is fixed. Each road has a transit time of zero time units and there is a traffic signal at the end of each road that has a cycle time of m time units and is *green* for exactly one time unit in each period. Waiting is only allowed at the traffic signals. The capacity of these m roads is not bounded, thus all demands may pass the signal during one period.

However, all roads lead to a narrow road e, which starts after the signals and ends at the destination. This road has a capacity of $u(e) = C$ flow units per time unit.

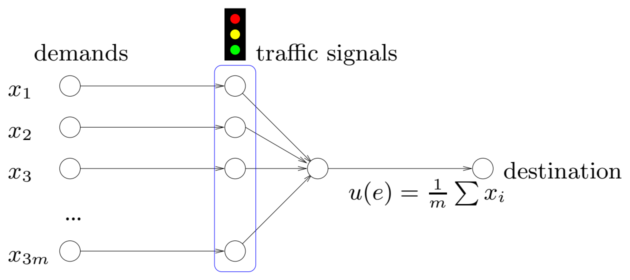

Figure 30: Traffic network for reduction from 3-Partitioning. The narrow road at the exits forces perfect coordination.

As the capacity of the narrow road cannot be exceeded, flow units will have to wait in front of the traffic signals when this small road at the exit is congested. If too much traffic signals turn *green* at the same time, flow units will even have to wait in front of *green* signals. More precisely, each signal is *green* for only one time step within the time horizon m. To make sure that no more than m time steps are needed for getting all flow units to the destination this one time step has to be used for sending all flow units of a commodity at once through the signal. Since $x_i > \frac{C}{4}$, no more than three commodities can be sent at once. Hence, exactly three signals have to be *green* at the same time step such that the capacity of the narrow road is exhausted.

Thus, if the related 3-Partition Problem has a Yes answer, we can use the exit road at maximum capacity all the time and we need m time units to empty the network. If the 3-partitioning has no Yes solution, then there will be congestion in the network and flow units will have to wait for the next *green* cycle yielding a higher total travel time. Consequently, the Signal Coordination Problem is at least as hard as the 3-Partition Problem. □

When allowing route choice and by considering related hard problems like Partition or Numerical 3-Dimensional Matching [66] even more realistic traffic networks than the one in the proof of Theorem 4.3 can be constructed and be used for the hardness proof. For example, one can find a planar network with node degree at most three for modeling partition by the help of signals and traffic (Figure 31). The main idea of the constructions is using traffic signals for sorting, assigning and splitting traffic. Yet, it should be pointed out that Partition (in contrast to 3-Partition) is weakly $\mathcal{NP}$-hard.

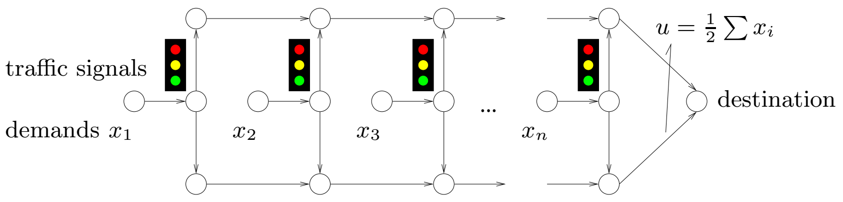

Figure 31: Traffic network for reduction from Partition. All road users at origin k start at time $t = k$. For each road, a free speed travel time of one time unit is assumed. All traffic lights have the same signal plan and four phases of equal length as follows: *green* for right turners $\rightarrow$ *red* $\rightarrow$ *green* for left turners $\rightarrow$ *red*. The narrow roads at the right end of the network limit the traffic to $u = \frac{1}{2}\sum_{i=1}^{k} x_i$ per time unit. If there is a partition of the demands into two sets of equal size, we can find a coordination which allows the road users to switch to the upper or lower road according to the partition without waiting at the traffic signals.

5 A New Model

5.1 Motivation

In this chapter, we will develop our combined model for traffic signal coordination and traffic assignment. We have already seen several models for at least one of these problems in Chapter 4. With these insights of advantages and disadvantages, we fix the key points that our model should provide, first.

The time-dependent behavior of traffic signals is one of the most important properties of our problem. Thus, our model has to capture this dynamic behavior. *Flows over time* are an adequate mathematical model that can be extended with traffic signals. Furthermore, this will allow mathematical programming techniques. Most practitioners do not concentrate on sound mathematical models, but they are mainly interested in good results that are applicable in practice. Most established models for signal coordination use genetic programming or other heuristic approaches. From a mathematician's point of view, this is not satisfying. For us, the guarantee of optimality or at least a measure of the quality of a solution is of the same importance as the solution value itself. Several approaches presented in Section 4.3 do not fulfill this requirement. Actually, none of the known models for simultaneous optimization of coordination and assignment provides a dual bound. Our model should improve this, i.e. it should be suitable for applying an exact mathematical programming approach.

In Section 4.3.1, we have discussed several possibilities for measuring the performance of coordinated signals and came to the conclusion that only total travel time which consists of transit times and delay, seem reasonable for the assignment problem. Thus, our objective is to minimize the total travel time in our combined model. Given a network $G = (V, A)$ with fixed capacities, transit times, commodities and traffic signals with fixed signal plans, we define the TRAFFIC SIGNAL COORDINATION TRAFFIC ASSIGNMENT PROBLEM (TSCTAP) to be the problem of finding a set of offsets and a traffic assignment, such that the total travel time is minimized.

Most likely, a traffic engineer would have to be convinced of a model without flow dependent travel times. The more cars are on a road, the more the average travel time increases. Classical link performance functions have this property, but they do not capture any time-dependent behavior like traffic signals or fluctuating flow values. Hence, some traffic experts think that link performance functions are not sufficient to model inner-city traffic [118]. Our model should capture such typical effects of traffic like platoons of cars. Further, we need a handy concept for travel times that does not disturb our exact mathematical programming approach. Therefore, we make the following simplifying assumption in our traffic model. The travel time on a link in the network splits into two components: the *free speed travel* time that is needed to bridge the distance of the link, and the *waiting time* in front of the intersection, i.e. in front of a traffic signal. In particular, we assume the free speed travel time to be independent of the load of the street. Although this assumption is not justified on highways or in rural areas, it is appropriate in inner-cities, where the distance between consecutive intersections is comparably small and a strict speed limit is present. The speed of a single car does not differ much from the speed of a platoon of cars. The waiting time mainly depends on the offsets of the subsequent traffic signals. This assumption will allow the formulation of flow dependent travel times in an implicit manner. We will thoroughly discuss the consequences of this assumption and its influence on system optima and user equilibria in this chapter.

Based on these assumptions, we properly define our problem:

Problem 5.1 (Traffic Signal Coordination Traffic Assignment Problem).
INPUT: *Network $G = (V, A, u)$ with capacities $u : A \to \mathbb{N}_0$ and travel times consisting of a constant transit time per arc, i.e. $t : A \to \mathbb{N}_0$, plus waiting time at the intersection; a set of traffic signals with arbitrary but fixed operating sequences at some of the intersections $v_n \in V$, $n \in \{1, \ldots, N\}$; a cycle time Γ that is the same for all traffic signals; commodities $\theta \in \Theta$, $\theta = (s_\theta, z_\theta, d_\theta)$ with origin s_θ, destination z_θ and demand d_θ.*
OUTPUT: *A dynamic flow and a set of offsets $\{\rho_1, \ldots, \rho_N\}$ which minimize the total travel time, i.e. the sum over the travel times of all road users.*

Now we have stated the basics of our input and output and can state the aims and anticipated properties of a model that we would like to get:

- Develop a model for simultaneous optimization of traffic signal coordination and traffic assignment. Consider feedback between coordination and assignment.
- Capture time-dependence:
  - dynamic travel times without static link performance functions,
  - traffic signals, and
  - fluctuating traffic, e.g. platoons.
- Objective: total travel time (transit time + delay).
- Realize flow dependent travel times via constant travel times plus flow dependent waiting times.
- If possible, approximate user equilibria with system optimal solutions.
- Solvability with mathematical programming. Provide dual bounds.
- Compute solutions of practical relevance, i.e. demonstrate applicability of the model with simulation results.

We will show that most of these aims and properties can infact be achieved by our new model. In the following, we first describe the various parts of this new model that are combined to solve the TSCTAP. These parts cover the cyclic time-expansion, expansion of intersections, and implementation of traffic signals. Building on these notions, we will give a mixed integer programming formulation, study basic properties, and discuss various subsequent improvements of the model. Afterwards, Chapter 6 is dedicated to the evaluation of our model with help of simulation tools.

Please note that some of the results were already presented at conferences [96, 97, 98].

5.2 The new model

5.2.1 Cyclically time-expansion

As explained earlier, for traffic signals and their coordination, a time-dependent model, capable of describing the time-offset between consecutive intersections, is a vital ingredient. Hence, flows over time and time-expanded networks seem to be the suitable mathematical model.

Yet, time-expanded networks are rather inefficient if the time horizon is large, since the number of time steps determines the number of network copies that have to be provided. However, since traffic signals have a periodical behavior it is not necessary to use a full time horizon expansion. Instead, we suggest a *cyclic time-expansion* where we expand only a time interval of size of the cycle time Γ and use only $k \in \mathbb{N}$ time steps of size $t = \frac{\Gamma}{k}$.

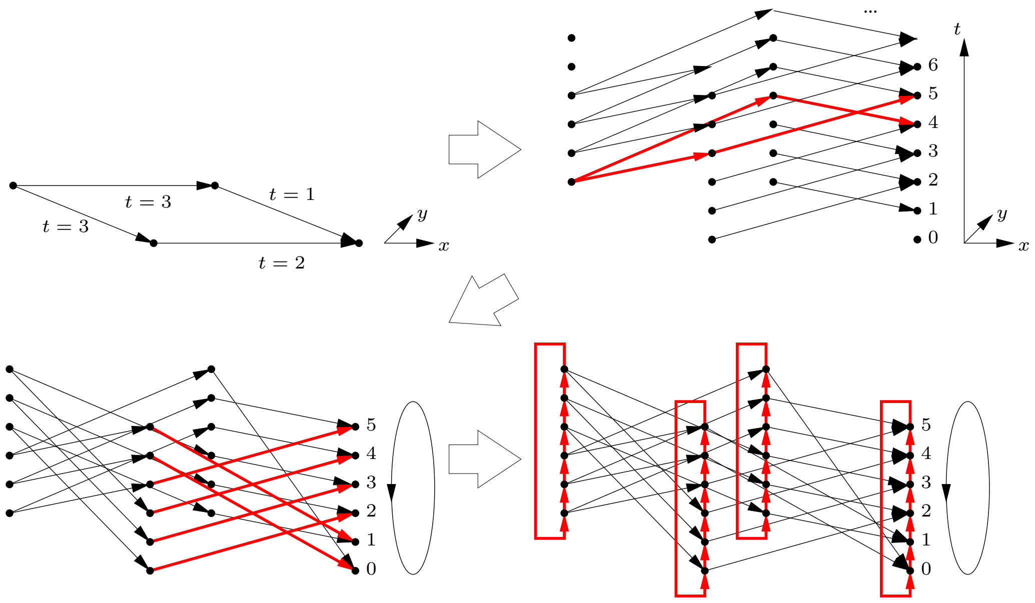

Figure 32: Simple example of a 4-vertex graph together with its time-expanded graph similar to the example in Figure 28 in Section 4.2.4. Again, the graph is embedded in the 2-dimensional Euclidean plane and the expansion is displayed along the third axis. In our model this time expansion is reduced to a cyclic time expansion with finitely many time steps. Waiting arcs allow buffering of flow units in a node for some time.

Furthermore, we add the arcs according to transit times modulo k with adjusted capacities. Of course, this treatment only applies to the indices of the incident nodes, the original transit time of an arc is preserved for further calculations. To model the ability to wait in front of an intersection, all copies of one node v are cyclically connected in chronological order with waiting arcs (v_i, v_{i+1}). Again, v_{k-1} is connected to v_0 which yields a *cyclically rolling horizon* (see Figure 32 and 34). A cost of one time unit is assigned to each waiting arc.

Please note that there is almost no difference between transit arcs and waiting arcs in the model. On waiting arcs one only moves in time whereas on regular arcs one moves in space and time. On all arcs the cost is the travel time on this arc. Thus, waiting time is travel time on waiting arcs. Hence, both kinds of arcs are treated in the same way and we do not need to explicitly distinguish between them in the modeling of traffic assignment.

5.2.2 Expanded intersections

To model intersections with different lanes, turning directions, and interior traffic signal offsets, we use a standard approach from traffic networks. Before the time expansion is employed, every intersection node is split up into several nodes for *incoming* and *outgoing traffic*; interior arcs connect the lanes (see Figure 33). Each of these arcs is assigned to one traffic light.

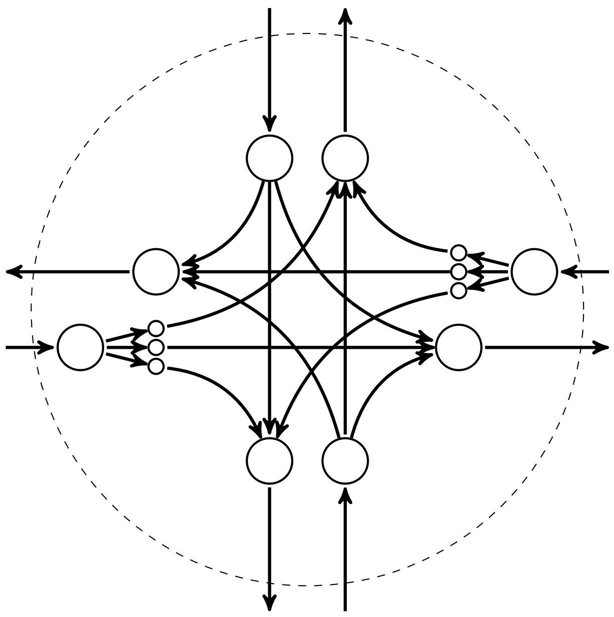

Figure 33: Expanded intersection with arcs for each turning alternative. The incoming nodes of the horizontal road are subdivided into three nodes to model different lanes and queues for the turning directions. The vertical road is smaller, thus, all cars are waiting in the same queue.

We will refer to the set of all interior arcs of intersections by $E \subset A$. By choosing appropriate capacities $u(e)$ and transit times $t(e)$, $e \in E$, these interior links form different *queues*, i.e. flow on the waiting arcs, for the several turning directions, because they limit the outgoing flow of the waiting arcs.

5.2.3 Modeling traffic signals

The traffic signals themselves can be modeled by binary decision variables that switch the capacities of the interior arcs at an intersection on or off, depending on the signal plan and the corresponding time step (see Figure 34). For each traffic light a matrix $Q^e \in \{0,1\}^{k \times k}$ is given; here ${Q^e}_{ij} = 1$ means that the particular traffic light is *green* at time step j when using offset $i\frac{\Gamma}{k}$. So each row stands for a certain offset and determines the operating sequence of the traffic light for this specific offset. For each interior arc e of the not time-expanded graph we create such a matrix. The signal plan is completely encoded in these matrices.

$$Q^e = \begin{pmatrix} 0 & 0 & 1 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & \cdots & 0 \\ \multicolumn{7}{c}{\dots\dots\dots\dots\dots\dots\dots\dots} \\ \multicolumn{7}{c}{\dots\dots\dots\dots\dots\dots\dots\dots} \end{pmatrix}$$

For each intersection n we introduce k binary variables $b^n = (b^n_1, \dots, b^n_k) \in \{0,1\}^k$ with constraint $\sum_{i=1}^{k} b^n_i = 1$ that describe the chosen offset at the intersection. More precisely, $b^n_i = 1$ is equivalent to offset $\rho_n = i\frac{\Gamma}{k}$ at intersection n. By multiplying this characteristic vector b^n with Q^e and the capacity $u(e)$ of a lane, we can switch the capacities of the links on or off and thus, represent the *green* and *red* lights.

Using $f(e) \leq b^n Q^e u(e)$ for all interior lanes of an intersection (where $u(e)$ is chosen in correspondence to the granularity of the time-expansion) we can map the complete dynamic behavior of traffic signals in our model.

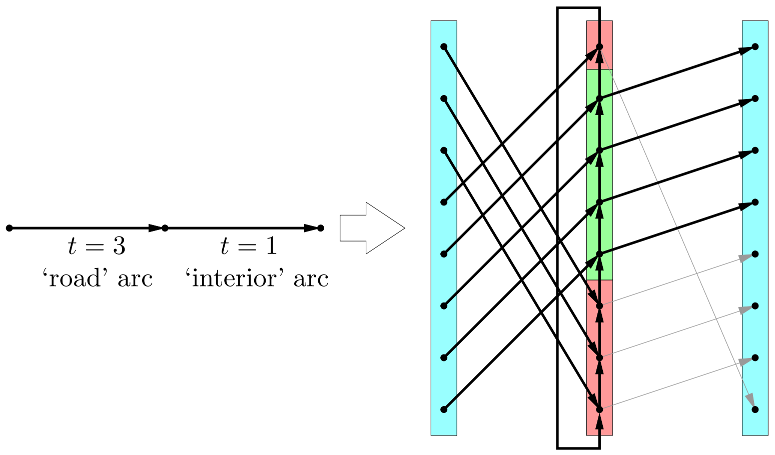

Figure 34: Cyclic time-expansion of a traffic signal. The gray arcs starting at the *red* phase of the signal are switched off and thus have zero capacity.

Putting together the various parts of the model we get the following definition of cyclically time-expanded networks.

Definition 5.2 (Cyclically time-expanded traffic network). *A cyclically time-expanded traffic network is a network, that is obtained from a traffic network $G = (V, A)$ by*

(i) expanding the intersections with arcs for turning alternatives and lanes,

(ii) cyclic time-expansion with respect to the cycle time, and

(iii) expanding signal plans.

5.2.4 Modeling traffic assignment

Building up on the cyclically time-expanded network, we can now consider the traffic assignment problem within this framework. Although flow can be considered to travel in a time-dependent manner through the cyclically time-expanded network, one should rather see this model as a static model that just captures some time-dependent aspects of traffic flow. Due to the cyclic repetition of the vertex and arc copies, a flow particle traveling through this network can be seen as a representative of a whole set of temporally repeated particles at every multiple of the cycle time.

On the other hand, there is a closely related interpretation of static network flow for traffic networks. In this interpretation one considers a flow-carrying path in the static network as a mapping of a corresponding amount of flow particles traveling over time through the traffic network at the corresponding flow rate. In other words, a flow-carrying path in the static network represents a constant rate of flow on this path in the 'real' time-dependent traffic network.

This interpretation suggests how to put together the two models, the cyclically time-expanded network on the one hand and the static traffic assignment model on the other hand. Basically, the demands for the different commodities have to be subdivided to the number of layers/time steps in the cyclically time-expanded network. The precise construction is given in the following.

Given (static) commodities $\theta \in \Theta$, $\theta = (s_\theta, z_\theta, d_\theta)$ in the original (static) network (sending d_θ units of flow from node s_θ to node z_θ), one has to extend them to the cyclically expanded network. Each demand has to be scaled to time Γ and can be divided uniformly among all copies of s_θ. The constraints for the sink nodes should be less restrictive, i.e., no uniform distribution is required. We propose to use a super-source and a super-sink, connected by a backward arc, as shown in Figure 35.

5.3 Mixed integer programming formulation

Let $G = (V, A)$ be a cyclically time-expanded traffic network and commodities $\theta \in \Theta$, $\theta = (s_\theta, z_\theta, d_\theta)$, capacities $u : A \to \mathbb{N}$, a set $E \subset A$ of interior arcs at intersections with associated matrices Q^e for each $e \in E$, travel times $t(e)$ for each link $e \in A$ and flow functions $f_\theta : A \to \mathbb{R}$ for each commodity. Now we can formulate a mixed integer program for TSCTAP.

We extend a common multicommodity min-cost circulation program for the cyclically time-expanded network by adding the binary variables and capacity constraints above.

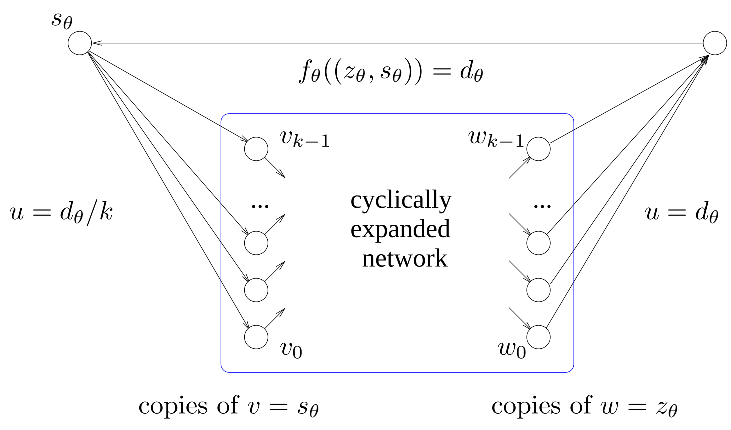

Figure 35: Demand splitting for commodities.

$$
\begin{array}{lllr}
\min & \sum_{e \in A} \sum_{\theta \in \Theta} t(e) f_\theta(e) & & \\
\text{s.t.} & 0 \leq \sum_{\theta \in \Theta} f_\theta(e) = f(e) \leq u(e) & \forall\, e \in A \setminus E & (18) \\
& \sum_{e \in \delta^+(v)} f_\theta(e) = \sum_{e \in \delta^-(v)} f_\theta(e) & \forall\, \theta \in \Theta \quad \forall\, v \in V & (19) \\
& f_\theta((z_\theta, s_\theta)) = d_\theta & \forall\, \theta \in \Theta & (20) \\
& \sum_{i=1}^{k} b_i^n = 1 & \forall\, n \in \{1, \ldots, N\} & (21) \\
& f(e) \leq b^n Q^e u(e) & \forall\, e \in E & (22) \\
& f(e) \geq 0,\; b^n \in \{0,1\}^k & &
\end{array}
$$

The constraints of type (18) fix the capacity bounds, type (19) implements the flow conservation and the constraints (20) force the circulation. Equation (21) ensures that exactly one offset is chosen at each intersection, and (22) permits flow only on arcs that are switched on with respect to the chosen offsets.

5.4 Properties of the model

First, we examine the assignment problem for fixed traffic signals.

Theorem 5.3. *Using the cyclically time-expanded network the traffic assignment problem for a fixed traffic signal coordination and for a fixed time granularity can be solved efficiently.*

Proof. If the traffic signal coordination is fixed, then all binary decision variables in our model are also fixed. Therefore, we obtain a standard linear program (LP) for the traffic assignment problem. Since the LP has polynomial size with respect to the input, i.e. the cyclically time-expanded network, it can be solved in polynomial time, e.g. by using the ellipsoid method [73]. □

Instead of using a standard LP solver for the assignment problem, one can exploit the network structure of the problem: as explained in Section 5.2.4, for each commodity artificial source and sink nodes can be added to create a circulation problem for these commodities with the help of a backward arc. One can apply this to all commodities and finally remove all arcs that are switched off by the decision variables. Now, any algorithm for the Minimum Cost Multicommodity Circulation problem can be used. Thus, we can also take advantage of fast combinatorial approximation algorithms (e.g. [67]) to solve the assignment even faster.

We can also observe an interesting property of the optimal solution, even for arbitrary signal coordinations.

Observation 5.4. *An optimal solution (system optimum) of the proposed multicommodity min cost flow problem is also a user equilibrium.*

Proof. Consider a flow with minimum total travel time, i.e. a system optimal flow. All transit times of arcs in the model are independent of the flow on that particular arc. Therefore, any route change of a single road user has no influence on the transit times of the arcs. By the optimality of the flow the new route cannot be shorter than the old one, since this would imply an improved system optimal solution, a contradiction.

By Wardrop's principle, a flow is in a user equilibrium, if the route choice of a single user does not improve on its travel time. Consequently, a system optimal solution of our model is also an user equilibrium. □

We confirm that this result holds if the cyclically time-expanded model is seen as a common static network. We compute a static assignment in a – although time is encoded indirectly – static network and we apply a result from static network flows. Furthermore, using a network with capacities user equilibria are not unique [37]. Thus, Observation 5.4 does not provide any statement about the relevance of this user equilibrium. In general, it is not guaranteed that this specific traffic assignment can be achieved in reality. Before we get back to the implications of Observation 5.4 in Section 5.5.3 and a dynamic interpretation, we have to study travel times in the cyclically time-expanded network first.

5.5 Analyzing link performance

So far, we have introduced a model with constant, flow independent transit times on links. This might seem too restrictive for a lot of practitioners in the first moment. However, there is infact a time-dependent behavior and it is captured in the time expansion. We will now try to uncover this time dependency by an analysis of our model on a single link with a traffic signal. Furthermore, we support the observations by a simulation of traffic on this link.

We will see that travel time on a link depends on numerous parameters, e.g. the distribution of incoming flow values over time. Furthermore, flow particles on the same link at different times will experience different travel times. We use *link performance* to term the average travel time on a link. We will not present a closed function for computing travel times. Nevertheless, we use the term *inherent link performance function* according to related traffic assignment models with flow dependent travel times. In the following, we fix most of the parameters and restrict the inherent link performance function to a small subset of its domain to at least demonstrate the most important properties of this implicit link performance in the cyclically time-expanded model.

5.5.1 Link performance of a single road

We consider a very small network which consists of a single road with a traffic signal in the middle similar to the network in Figure 34. Assume a set of flow values assigned to the network and a fixed coordination to be given. The total waiting time in this small network can be determined by multiplying flow value and transit time and summing over all waiting arcs. Increasing flow will increase the waiting time, because more flow will be assigned to the waiting arcs. Furthermore, due to the bounded capacities, the flow units on the waiting arcs may not leave completely on the first *green* outgoing arc if the arc does not provide enough capacity. Instead, the flow will have to use more waiting arcs until the accumulated flow is drained off. This relation is illustrated in Figure 36. Therefore, if the incoming flow is raised linearly on all copies of an arc, then the growth of the waiting time will be not linear but rather quadratic. More precisely, the obtained function is piecewise linear, but converges to a quadratic function if the length of the timesteps in the expansion converges to zero.

In Figure 37 we present the relation between flow and average waiting time on a single link in the cyclically expanded network, i.e. our network consists of only one road with a traffic signal. The traffic is equally distributed on all copies of this link, a traffic signal with a cycle time of 60 seconds and a *red* time of 20 seconds is put at the end of the road and a free speed travel time of 10 seconds is assumed. Additionally, a capacity reduction from two lanes to one lane at the traffic signal was used. We now compute the average travel time on this link with respect to the flow value. The obtained travel time in Figure 37 demonstrates the capability of our waiting arc model: although using only constant travel times the inherent, implicit link performance functions of the model are not linear. Even better, they seem to resemble common standard link performance functions (cf. Figure 26).

Please note that Figure 37 visualizes the average travel time. The individual travel time of a single flow particle depends on its arrival time at the traffic signal. It ranges from 10 seconds for flow units arriving at *green* with no waiting queue at the signal up to 30 seconds for particles arriving at the beginning of the *red* phase. Further, we can also compute the *expected* free speed transit time. With a probability of $\frac{2}{3}$ we arrive at the traffic signal at *green*. Thus, we arrive at *red* with a chance of $\frac{1}{3}$ and the waiting time is equally distributed between 0 and 20 seconds in this case. Thus, the expected free speed travel time is $10 + \frac{1}{6} \cdot 20 = 13\frac{1}{3}$ seconds.

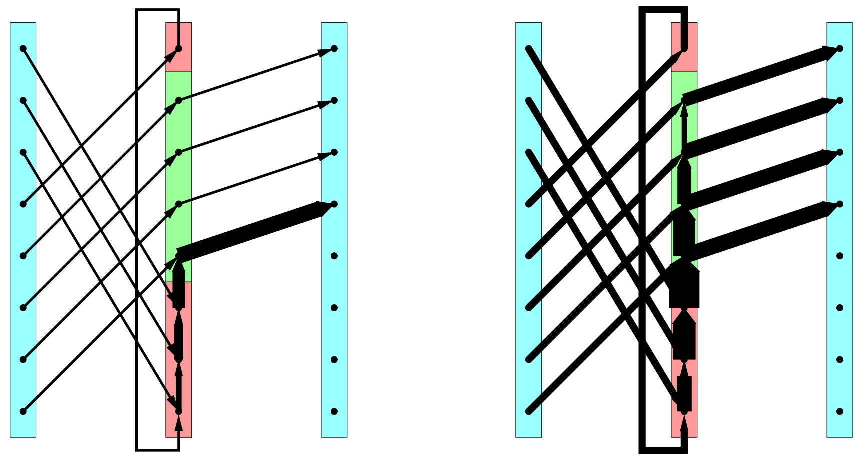
Figure 36: Accumulation of waiting time. The thickness of the arcs depicts the flow value. On the left side, incoming flow is low. All flow accumulated during the *red* phase can leave the signal directly when it becomes *green*. On the right side, the flow is tripled. Thus, much more flow is accumulated on the waiting arcs. Furthermore, due to the capacity bound of the outgoing street, more waiting arcs have to be used. Waiting time increases in this scenario by a factor of 4.8.

This analysis suggests that our model can capture flow dependent travel times similar to standard link performance functions. However, we assumed traffic to be equally distributed over time. This is rather unrealistic for inner-city traffic. In particular, traffic signals create platoons of cars. Thus, we extend our analysis to flows with quickly changing traffic density.

5.5.2 Link performance for platoons of cars

In the cyclically time-expanded network *platoons* can be modeled by different flow values on the copies of each particular arc. Some arcs may be used at full capacity, other arcs may not be used at all. Varying flow values can be interpreted as platoons of different lengths and densities. Thus, our model is capable of creating, splitting, merging, compressing and stretching these platoons.

Let us return to the single road from above. Even in this small example, a platoon can change the characteristics of the average waiting time dramatically. A platoon of cars can be described by its length and its density. For simplicity, we fix the density to be constant, thus the platoon length can be considered to be proportional to the average link flow.

To visualize the occurring effects even better, we change the *green* time to 30 seconds and the capacity of the outgoing road is set to twice the capacity of the incoming road. The average travel time now depends on two parameters: the length of the platoon and the arrival time of the head of the platoon at the traffic signal. In Figure 38, the calculated average travel time is plotted versus the platoon length. On the left side, the

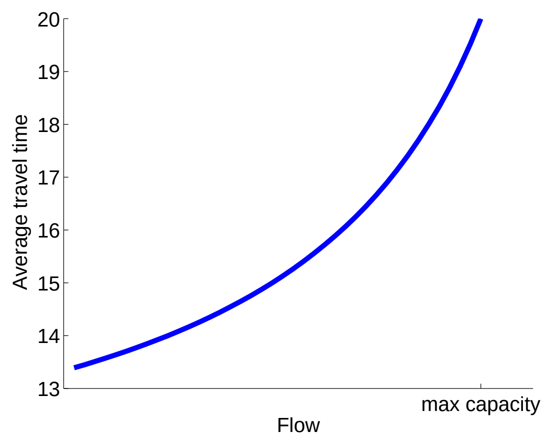

Figure 37: Computed average travel time with respect to flow on a single link in the cyclically time-expanded network. Incoming traffic is equally distributed over time.

first flow unit arrives at the intersection 10 seconds before the signal turns *green*. On the right side, the first flow particle arrives 20 seconds after the signal turned *green*. The different characteristics show the influence of these two offsets on our travel times.

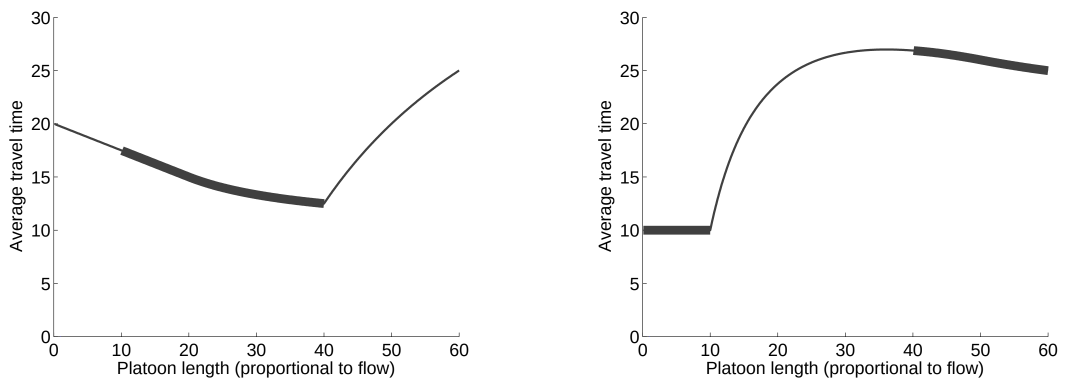

Figure 38: Inherent link performance function for a platoon of cars in the cyclically time-expanded model. On the left side, the platoon is arriving 10 seconds before the signal turns *green*. Due to the higher capacity of the outgoing road the traffic signal can be used to densify the platoon, e.g., the cars may leave on parallel lanes. The first car of the platoon has to wait for the longest time, the last cars may pass the signal without stopping. On the right side, the first flow particle arrives 20 seconds after the signal turned *green*. Thus, small platoons can pass without stopping, long platoons are splitted by the traffic signal which turns *red* 10 seconds after the head of the platoon has passed.

Obviously, the implicit travel times in our model are quite different from standard static link performance functions found in traffic literature [46, 69]. There, link performance functions are assumed to be convex and monotonically increasing. This simplifies the analysis of user equilibria considerably. In contrast, the inherent link performance functions in the cyclic time-expanded model may be decreasing or concave on some intervals (see Figure 38). However, this is no inaccuracy or disadvantage of our model. Anticipating the simulation results in Chapter 6, these characteristics of travel time functions can be reproduced by state-of-the-art traffic simulation tools (see Figure 39) which supports our model.

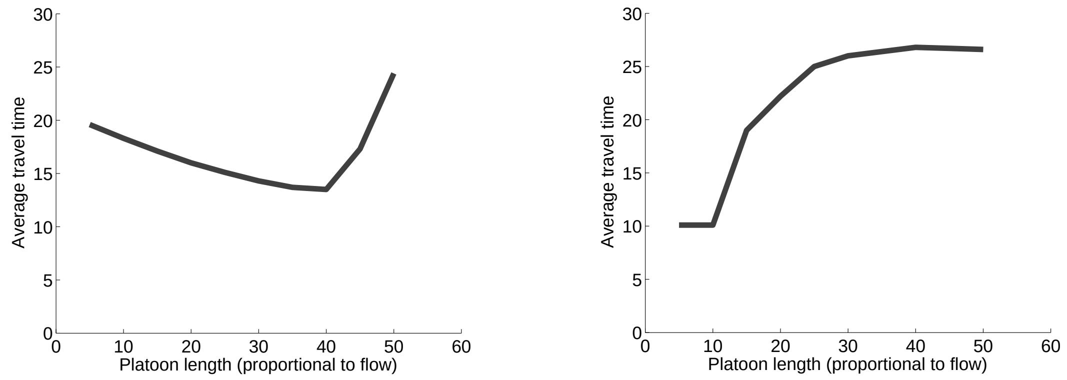

Figure 39: Link performance for the same scenario as in Figure 38, but now simulated and measured with VISSIM. Note that only the average travel time is displayed. The actual time depends on the position of the car within the platoon. Comparing to Figure 38, the predicted travel times fit remarkably well to the simulated travel times even without a careful calibration of our model.

Finally, the average travel time of a flow particle on a link in the cyclically time-expanded model is depicted in Figure 40 with respect to both parameters platoon length (i.e. traffic flow) and arrival time (i.e. offset of the signal). Hence, one can conclude that the common assumptions on static link performance functions may be realistic in rural areas or on highways. But they are not accurate enough for signalized inner-city traffic. Further, it can be concluded that the cyclic time expansion is able to model both classical link performance and link performance for traffic signal coordination and platoons.

5.5.3 Link performance in networks and user equilibria

We have seen that the inherent link performance of the cyclically time-expanded model is flow-dependent. On the one hand, this is not really surprising. Even in minimum cost flows the average cost per unit shipped from the source to the sink increases with increasing demand. When the capacity of the cheapest path is reached, more expensive paths have to be used for the remaining flow.

On the other hand, applying Wardrop's principles and classical game theory, we can conclude that the system optimum of our cyclically time-expanded model is also a user

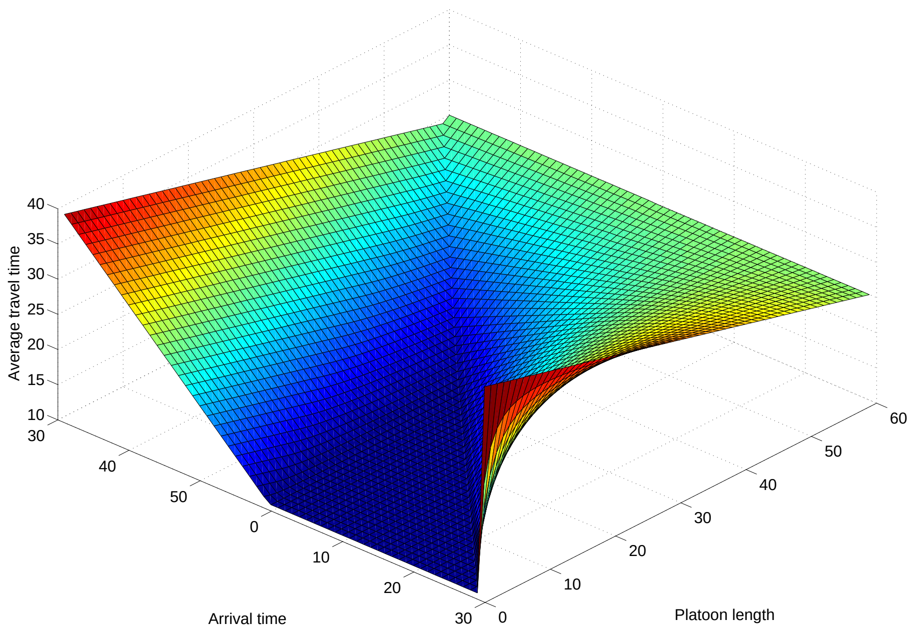

Figure 40: Calculated average travel time for a road user in a platoon with respect to platoon length and arrival time at the traffic signal. The signal turns *green* at $t = 0$ and *red* at $t = 30$. The inherent link performance functions of Figure 38 are obtained as profile for $t = 50$ and $t = 20$. Note that for fixed platoon length one obtains waiting time functions very similar to those used by Wünsch et al. [95, 115].

equilibrium (Observation 5.4). Does this result allow for effects, e.g. Braess's paradox, that are linked to flow dependent travel times?

First of all, Observation 5.4 is not a contradiction to classical game-theoretic results. As shown by Correa et al. [37], the value of user equilibria in a network with capacities is not unique. Thus, several equilibria may occur in the cyclically time-expanded network.

In the classical assignment with static link performance function, each flow particle on a link experiences the same travel time on this link. If an additional flow unit is assigned to a path containing this link, then this increases the travel time of all flow particles, including the new one, by the same amount. In the case of capacitated links, the saturation of some arcs restricts the route choice of the remaining flow units. This is the main reason for the existence of multiple equilibria. According to the definition of capacitated user equilbria (cf. [37]), only unsaturated paths are considered for an alternative routing. A flow particle can only change its path through the network if all links of the new path have sufficient remaining capacity. Thus, two flow particles crossing twice may both improve by swapping the intermediate part of their routes. Unfortunately, co-operation is not allowed and they may block each other.

Considered statically, this applies one-to-one to the arcs and waiting arcs of the cyclically time-expanded network. In contrast, the dynamic point of view in the cyclically time-expanded model provides a new concept and interesting questions arise. Considering all copies of an arc in the cyclically time-expanded network together with the waiting arcs as a single link, flow particles may experience quite different travel times depending on their arrival time at the origin node. Further, we may apply the saying *first come first serve*. Assume an additional particle is assigned to a link. If the flow particle arrives earlier at the link than other flow units, it may simply jump the queue and push in at the first position. Thus, it experiences the smallest travel time among all flow units and all other particles are slowed down.

Now, we can also interpret the user equilibrium of Observation 5.4. In our system optimum no road user can switch to another road to improve its travel time without jumping into any queue. Using an unsatured path corresponds to always being the last car in each queue on this path. Thus, we may think of our road users as *friendly* or *pessimistic*[10]. They are friendly, because they join the queue at the end and they do not push into queues. On the other hand, it seems they calculate the travel time of an alternative route pessimistically by assuming to be always the last car in each queue. Unfortunately, this interpretation also implies that this user equilibrium is of minor practical relevance. Due to the traffic signals, road users are pushed right into the queues and traffic is getting mixed up. The pessimistic point of view over-estimates the travel times on alternative routes.

Concluding, for any user equilibrium we have to analyze whether there is a flow particle that could improve its travel time by switching to another path with jumping into queues permitted. We may speak of a *fair* or *realistic* user equilibrium if no such flow particles exist. Such an equilibrium is most likely hard to compute exactly. It would imply an assignment where a chronological ordering of the paths is defined. However, some flow units may arrive at a certain node at the same time. One would still need a policy to handle these situations. Even more complicating, we have dissolved the temporal order in the cyclically time-expansion. There is no chance to determine, whether any event happens before or after any other event. The concept of user equilibria in dynamic traffic assignment is not applicable. Thus, we have to leave the evaluation of user equilibria as an open question here. Fortunately, referring to the simulation results in Chapter 6, the system optimum solution already provides improved coordinations of practical relevance.

Finally, we support the explanations above by a small example. The scenario is similar to the Braess's paradox in Section 4.2.3 and the underlying graph is displayed in Figure 41. The network consists of 4 nodes and 5 arcs. Each link has a capacity of $u \equiv 1$. The links (v_1, v_2), (v_2, v_3), and (v_3, v_4) have a transit time of 0 seconds, whereas the other links have a transit time of 20 seconds. Traffic signals are installed in the nodes v_2, v_3 and v_4 with a cycle time of 60 seconds and a *green* time of 30 seconds for each traffic light. All intersections have two turning directions and we choose interior offsets of 0 and 30 seconds for the beginning of the *green* phases, respectively. Additionally, there is one commodity from v_1 to v_4 with a demand of d units of flow per second.

[10]This decision is left to the reader.

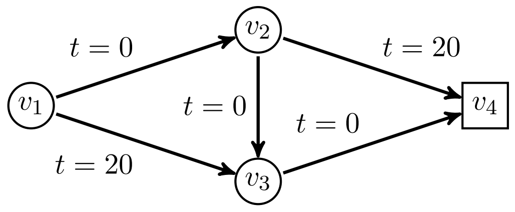

Figure 41: Small example of link performance in a network. Arc labels correspond to the free speed travel time on the links in seconds. All links have the same capacity $u \equiv 1$. All traffic signals are *green* for 30 seconds.

Braess's paradox is originally a static network flow. If we consider the above scenario in a static setting without capacities and disregarding the traffic signals, nothing surprising will happen. With constant travel times, all flow particles choose the fastest path via v_2 and v_3. However, flow will use the paths $((v_1, v_2), (v_2, v_4))$ and $((v_1, v_3), (v_3, v_4))$ if we introduce capacities and the capacity of the shortest path is exceeded.

Now consider a dynamic flow with capacities, still without traffic signals. Assume a certain demand of flow units, all of them willing to start at time 0. In the first 20 seconds, the path via v_2 and v_3 is used at full capacity, since there is no faster possibility to reach v_3. After this initial time, route choice depends on the right of way in node v_3. If the road users arriving from v_2 have the right of way, then they will block the path $((v_1, v_3), (v_3, v_4))$. All flow particles will use the fastest route, but some of them will have to wait at v_1 for a long time. If traffic participants coming from v_1 have the right of way in v_3, then this path will be used. A flow unit could wait for more than 20 seconds at v_1 to use the fastest path or it could start directly traveling to v_3. With the right of way, this will be its fastest connection. For symmetry reasons, path $((v_1, v_2), (v_2, v_4))$ is also used with the same flow henceforward. Both possibilities are user equilibria in the terms of Wardrop. Pushing other flow particles to other paths is not allowed. However, only the second equilibrium is indeed system optimal.

With traffic signals, the right of way at v_3 changes periodically. Each of the two equilibria is given preference alternately. Consequently, the assignment may switch between both solutions. In Figure 42 we present the system optimal flow pattern for a 50 % utilized capacity, i.e. $d = 0.5$, and incoming traffic equally distributed over time. The model suggests a perfect coordination for the fast path via v_2 and v_3. However, half of the flow particles arrive at v_1 at *red* at the corresponding lane. Those who arrive shortly after the beginning of the *red* phase choose the alternative route to avoid waiting. The other flow units queue up for using the fast route.

The cyclically time-expanded model suggests a quite different solution when traffic increases. For $d = 1$, both paths $((v_1, v_2), (v_2, v_4))$ and $((v_1, v_3), (v_3, v_4))$ are used, which is indeed the sole possibility to handle the demand without exceeding capacities. Surprisingly, the former quick connection via (v_2, v_3) is assigned a 'red wave' automatically. Now, this is the slowest path between v_1 and v_4 and the system optimum is a stable user equilibrium in this case.

With an appropriate choice of traffic signal plans and fixed offsets, we may also con-

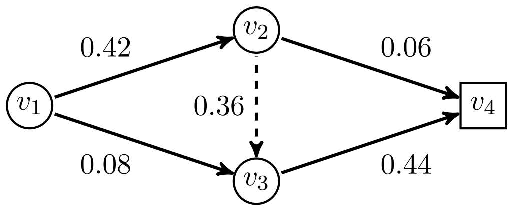

Figure 42: Optimal solution for the small example of Figure 41 with 50 % load. The arc labels correspond to the total flow on the link copies in the cyclically time-expanded model. The fastest road via v_2 and v_3 is given an additional preference by a perfect coordination.

struct system optimum assignments that are not stable user equilibria. Let us split v_1 into two nodes which admits two commodities with a fixed ratio of demands. If we choose the parameters such that the path via v_2 and v_3 is only slightly faster than $((v_1, v_2), (v_2, v_3))$, selfish flow particles of the first commodity will switch to this route. However, they will block the second commodity traveling via v_3 at the signal at v_4, if waiting is necessary there. A system optimal solution would prefer the upper path for the first commodity.

Nevertheless, the example provides another observation. Metaphorically speaking, our road users are very clever and they use the coordination at its limit. A driver knows whether she will miss the *green* phase at a traffic signal several intersection in advance and her route choice depends on this knowledge. In our small example, already the arrival time at the first traffic signal provides enough information for a qualified choice between three routes. This seems unrealistic for real world problems, since traffic is never completely constant and depends on several stochastic influences. This effect may be reduced by a proper calibration of the model, e.g. by decreasing capacities or *green* times. Furthermore, flow particles can wait everywhere in the network, even in front of *green* traffic lights. Additionally, they may wait such that other particles can overtake. This happens when flow has to wait somewhere on its path anyway, but the passing particle has to keep in time with a green wave on its path. We discuss this *first-in first-out* property in Section 5.6.5.

5.6 Model improvements

In this section we propose some refinements and extensions of our model that allow for an even more realistic modeling of road traffic in signalized networks.

5.6.1 Split and phase optimization

The first extension of the cyclically time-expanded model addresses split times and phase order. Until now, we only considered offset optimization and assignment. However, changes in assignment lead to different demands at single traffic lights. Thus, adjusting split times could improve the capacity of an intersection. Optimizing the phase order of large junctions with many lanes may also improve the coordination significantly.

In contrast, the fixed interior offsets and split times of the present model are an important advantage. The signal plans in the matrices Q^e can be designed by a traffic engineer who takes all safety aspects into account. If we try to optimize split times and phase sequences, we have to take care of these rules. Hence, we will need much more variables and constraints to ensure the interior logic of traffic lights at a single intersection. We concentrate on the following constraints:

- Crossing directions must not have *green* at the same time.
- Some directions must have *green* at the same time.
- There has to be enough time to clear the intersection at phase changes before other any other direction gets *green*.
- There has to be a minimum duration of each *green* and *red* phase.

In reality, many more constraints can occur, e.g. due to pedestrians. For simplicity, we only discuss a small standard intersection with 2 phases as depicted in Figure 43. The common cycle time is Γ and we use a cyclical expansion with k time steps. Thus, in the following, all indices apply modulo k.

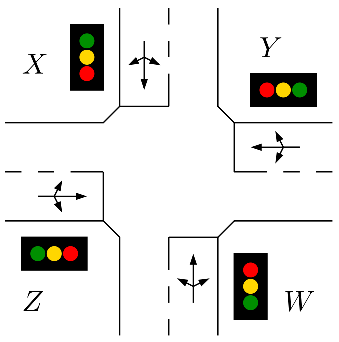

Figure 43: Simple intersection with four signal groups W, X, Y and Z.

The intersection n is expanded as usual. In contrast to the present model, we introduce k binary variables $b^n_W = (b^n_{W,1}, \dots, b^n_{W,k})$ for each signal group that represent the status of the signal directly. We refer to these variables as *status variables*. $b^n_{W,i} = 1$ implies that signal group W at intersection n is *green* at time step i. Otherwise, the signal is *red*. These binary variables switch the capacities of the corresponding interior lanes directly, i.e. we introduce constraints $f(e_i) \leq b^n_{W,i} u(e_i)$ for each interior arc of the cyclically time-expanded intersection. To keep the variables in a readable format, we will omit the index n in the following.

Thus, we bypass the matrices Q^e, but the signal can change in arbitrary sequences. Therefore, we demand that the signal W switches from *red* to *green* and from *green* to *red* only once during one cycle. Again, we use $2k$ binary variables $b^{\text{on}}_W = (b^{\text{on}}_{W,1}, \dots, b^{\text{on}}_{W,k})$

and $b_W^{\text{off}} = (b_{W,1}^{\text{off}}, \ldots, b_{W,k}^{\text{off}})$, respectively. This variables act like the decision variables for the offset in the present model and we refer to them as *decision variables*. Thus, $b_{W,i}^{\text{on}} = 1$ means that signal group W switches from *red* to *green* at time step i. The following constraints fix the number of switching operations per cycle to 1.

$$\sum_{i=1}^{k} b_{W,i}^{\text{on}} = 1 \text{ and } \sum_{i=1}^{k} b_{W,i}^{\text{off}} = 1$$

Now, we link the decision variables to the status variables. Remember to regard time modulo k.

$$b_{W,i} - b_{W,i-1} \leq b_{W,i}^{\text{on}} \ \forall i \in \{0, \ldots, k-1\}$$

Thus, if $b_{W,i}^{\text{on}} = 1$, then the signal may switch to *green*, i.e. $b_{W,i-1} = 0$ and $b_{W,i} = 1$. If $b_{n,W,i}^{\text{on}} = 0$, then $b_{W,i-1} = b_{W,i}$, that is the status is not changed. Similarly, we formulate the constraints for switching the signal to *red*.

$$b_{W,i} - b_{W,i-1} \geq -b_{W,i}^{\text{off}} \ \forall i \in \{0, \ldots, k-1\}$$

However, it is not guaranteed that the signal switches at all. We combine this with a constraint for minimum *green* and *red* times. If the signal should be *green* for at least g time steps, we require:

$$\sum_{i=1}^{k} b_{W,i} \geq g.$$

If the signal should be *red* for at least r time steps, we require:

$$\sum_{i=1}^{k} b_{W,i} \leq k - r.$$

Therefore, if the signal is *green* for at least one time step and it is *red* for at least one time step, two switching operations are forced.

So far, we can switch traffic lights and we can control capacities. We continue with the safety constraints. Crossing directions must not have *green* at the same time. However, *red* for both directions is possible. Obviously, the following constraints ensure safe crossing for two signal groups W and X (cf. Figure 43).

$$b_{W,i} + b_{X,i} \leq 1 \ \forall i \in \{0, \ldots, k-1\}$$

Furthermore, we have to guarantee a clearance time. Together with the previous inequality, a clearance time of at least one time step is forced by the following constraint.

$$b_{W,i}^{\text{off}} + b_{X,i}^{\text{on}} + b_{X,i+1}^{\text{on}} \leq 1 \ \forall i \in \{0, \ldots, k-1\}$$

Thus, if signal group W switches to *red* in time step i, then X cannot switch to *green* at the same time step or at time step $i+1$. Longer clearance times are possible. One may even insert another phase. Hence, clearance time constraints have to be formulated for each pair of traffic lights. In contrast, we may also fix the clearance time and the phase order with the next equation if desired.

$$b^{\text{off}}_{W,i} = b^{\text{on}}_{X,i-2} \ \forall i \in \{0, \dots, k-1\}$$

Finally, if two signals should switch to *green* at the same time, e.g. group W and Y in Figure 43, we may use the following constraints.

$$b^{\text{on}}_{W,i} = b^{\text{on}}_{Y,i} \ \forall i \in \{0, \dots, k-1\}$$

Summarizing, we have introduced several linear equations or inequalities for the most important safety constraints at a traffic signal. Their mode of action is mostly obvious. Furthermore, it is easy to create a mixed integer program similar to the MIP presented in Section 5.3 for a complete cyclically time-expanded network.

This MIP enables to optimize the signal plans and the traffic assignment simultaneously. Thus, offsets are optimized implicitly, since they are encoded in the new decision variables.

Instead of k binary offset variables per intersection, we have $3k$ binary variables per traffic light/lane. Hence, the MIP becomes large in practice and it is probably more difficult to compute an exact solution.

5.6.2 Multiple signal plans

Traffic signals are often capable of switching between different programs or modes, e.g., different *red-green* splits. Our model can be extended to the case of several programs or modes by adding these programs to the matrices Q^e. Thus, the binary variables not only choose the offset but also the operating sequence. This provides a trade-off between the present model and the extension in Section 5.6.1.

5.6.3 Flow dependent travel times with fan graphs

As pointed out above, waiting arcs are a very useful tool to model flow dependent transit times at traffic signals. In the literature, flow dependent travel times on long roads are also motivated by the interaction between moving cars. So far, this is not captured by our model, since we assume constant transit times on a link. For more realistic flow-dependent travel times we can make use of a model used for flows over time. In [93] a fan-like time-expanded network is suggested that models different travel times for different flow-rates on a given arc. Consider two intersections v and u with free-speed travel time τ on $a = (v, u)$. Instead of only adding one arc $a_t = (v_t, u_{t+\tau})$ for each time step of the expansion, several additional arcs $a_{t,i} = (v_t, u_{t+\tau+i})$ are added and the capacity is split among those copies. See Figure 44 for a visualization of such a fan. Therefore, in a situation with low load the original arc a_t provides enough capacity and all road users can be routed with the free speed travel time. Considering a rush-hour scenario some flow units are assigned to the 'slower' copies $a_{t,i}$ yielding an increasing average travel time. Note that this flow dependency does not touch the conclusion of Proposition 5.4.

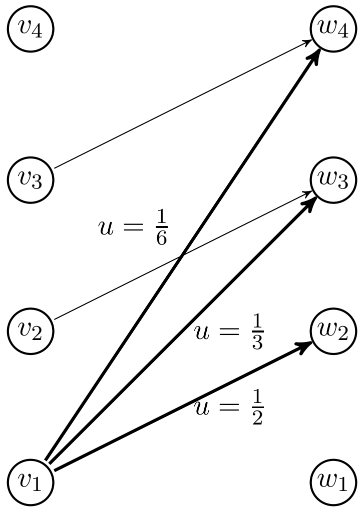

Figure 44: A fan-like expansion to model time-dependent travel times. Note that the capacities are not equally distributed.

5.6.4 Queues

Earlier, we introduced waiting arcs (v_i, v_{i+1}) between consecutive copies of a node v, but up to now the capacities of these arcs have not been discussed. For modeling the restriction of small streets, we propose choosing the capacity with respect to the length of the street. This bounds the queue at a traffic signal and allows backing-up of traffic over several consecutive intersections.

In particular, this may lead to traffic users waiting in front of *green* traffic signals. Flow conservation propagates the limited capacity of the outgoing link and its waiting arc to the incoming links of an intersection automatically. Thus, cars cannot pass a signal when the outgoing road is blocked and *green* time elapses unused.

Furthermore, we introduced waiting arcs at all nodes in the network. However, a queue directly behind the intersection is unrealistic. In the following, we set the capacity of waiting arcs between nodes for outgoing traffic to zero. The capacity of waiting arcs between nodes for incoming traffic is limited as above.

5.6.5 First-in first-out

Up to now, the queues of our model do not provide the *first-in first-out* property. Thus, road users may overtake each other.

Overtaking in the queues can be reduced with help of confluent flows. Flow carrying path of a confluent flow cannot cross each other (cf. Chapter 2). This can be used on the waiting arcs to disable passing. To achieve this, waiting arcs are expanded to a *confluent waiting net*.

In detail, we also expand a node, where waiting is possible, in direction of the queue. Let $v \in V$ be a node in the unexpanded network. In the time expansion with k time steps, we create copies $v_1, \dots, v_k$. Now, each node v_i is split into k copies $v_{i,1}, \dots, v_{i,k}$. We add arcs:

- from $v_{i,j}$ to $v_{i,j+1}$ $\forall i \in \{1, \dots, k-1\}$ $\forall j \in \{1, \dots, k-1\}$,
- from $v_{i,j}$ to $v_{i+1,j+1}$ $\forall i \in \{1, \dots, k-1\}$ $\forall j \in \{1, \dots, k-1\}$,
- from $v_{i,j}$ to $v_{i+1,j}$ $\forall i \in \{1, \dots, k-1\}$ $\forall j \in \{2, \dots, k-1\}$.

Indices i apply modulo k. This yields the cyclically time-expanded *waiting net*. On this part of the network, we demand confluent routing at all nodes $v_{i,j}$, $i = 1, \dots, k$ and $j = 2, \dots, k$ for all commodities concurrently. This *confluent waiting net* is connected to the remaining cyclically time-expanded network. Arcs for incoming flow end in $v_{i,1}$ with travel times analogically to the cyclical time expansion. Similarly, arcs for outgoing flow start in $v_{i,k}$. The construction is displayed in Figure 45.

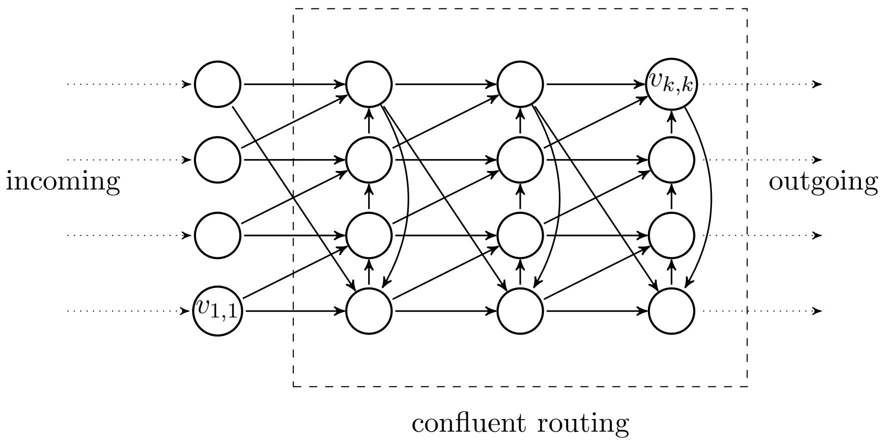

Figure 45: Example for a confluent waiting net for $k = 4$ time steps. We do not need a capacity constraint for single arcs of the waiting net, since flow is bounded by the outgoing arcs and the confluent conditions. The length of the queue (compare Section 5.6.4) can be limited by bundle constraints on the total flow at a certain time step.

Flow on the arcs is bounded by the capacity of the outgoing arcs and the confluent routing conditions. Since flow cannot split, the capacity constraints of the outgoing arcs automatically assign to every arc in the confluent waiting net. Confluent routing also permits crossing of flow. Note that incoming flow may split once before the confluent routing starts. Thus, flow can be grouped to fulfill the capacity constraint of the outgoing road exactly, but cars cannot overtake each other. The order is preserved, a *first-in first-out* queue is achieved.

On the other hand, this is only a theoretical approach. Most confluent flows problems are $\mathcal{NP}$-hard. Thus, we cannot hope to solve the coordination problem with a confluent subproblem efficiently.

5.6.6 Intersections without signals

Not all intersections in an urban area are equipped with traffic signals. We handle these junctions as follows.

The intersection is expanded as usual. There exist no binary variables for switching capacities on or off. Instead, we use bundle constraints for the capacity bound, i.e. the capacity is shared between crossing lanes. In detail, if e_1 and e_2 cross each other, we demand $f(e_1) + f(e_2) \leq \max\{c(e_1), c(e_2)\} \; \forall i \in \{0, \ldots, k-1\}$. This reduces the capacity of the intersection significantly. Furthermore, if there is a lane with right of way, we delete the waiting edges for this direction. Thus, the corresponding traffic participants cannot wait. Consequently, road users in the other direction have to give way.

6 Experiments

In this chapter we evaluate the applicability of the presented model by discussing various simulation results. We apply two well-established microscopic traffic simulation tools: MATSim (TU Berlin) and VISSIM (ptv AG) for our studies on four different scenarios.

Furthermore, we analyze runtimes for solving the mixed integer program of our model with CPLEX. We discuss the influence of several parameters and try to improve the performance.

6.1 Scenarios

Real-world data for offset optimization is always hard to get. One needs detailed information on the network, the traffic flow, and the signalization. Furthermore, we do not just need load or usage information on the roads, but rather commodities with origin-destination information. Additionally, we need geometric information about the position of traffic lights, lanes etc. as input for the simulation tools.

Fortunately, we were able to collect the detailed data for four real-world scenarios with help of several research partners.

- Braunschweig, Germany, 10 signalized intersections
- Cottbus, Germany, 32 intersections
- Denver, Colorado, 36 intersections
- Portland, Oregon, 16 intersections

6.1.1 Portland and Denver

Our research partner ptv AG kindly provided us with VISSIM-models for Portland and Denver. We could extract the relevant data out of these models. Furthermore, Wünsch [154] also uses parts of the inner-city networks of these two North American cities to evaluate his platoon model. The author would like to thank Gregor Wünsch for providing his results which admit the opportunity of benchmarks.

With a population of about 570,000 (2007), the city of Portland is the largest city in the state of Oregon and one of the largest cities in the Pacific Northwest. However, we only consider a small part of downtown Portland which is depicted in Figure 46. The 16 intersections of the chosen network between *Broadway* and *4th Avenue* and between *Taylor Street* and *Madison Street* are equipped with pretimed signals with a cycle length of 54 seconds. *Green* times reach from 17 up to 31 seconds. All streets are one-way with a varying number of lanes. Wünsch [154] identified 13 commodities.

We will use the Portland scenario to compare our model with the present coordination, a coordination obtained with TRANSYT, and the platoon model of Wünsch. Restrictively, route choice is nonrelevant in this scenario. Most of the given commodities will use the shortest paths to their destinations. The one-way streets lead to long detours

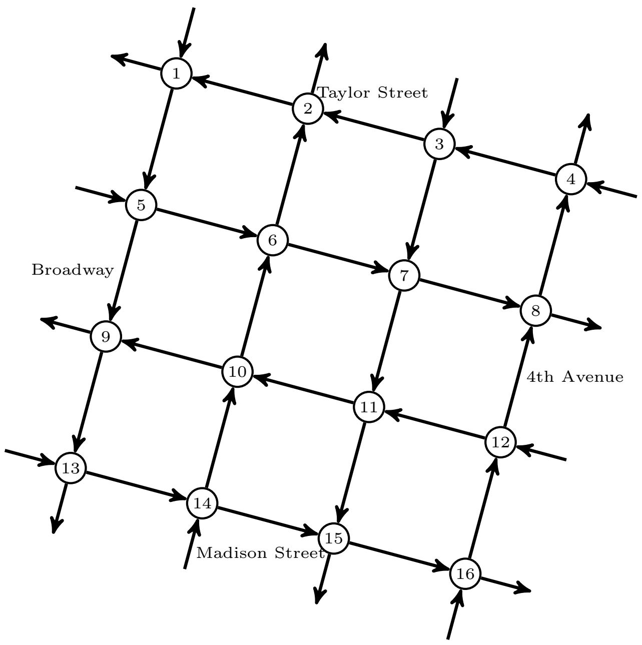

Figure 46: Network of downtown Portland, Oregon. The square between *Broadway* and *4th Avenue* and between *Taylor Street* and *Madison Street* consists of 16 intersections with pretimed traffic signals and alternating one-way streets.

if one decides to leave the direct path. Thus, our model will not be able to show its capability of simultaneous traffic signal optimization and traffic assignment.

As a similar example we investigate the inner-city of Denver. Located in the foothills of the Rocky Mountains, Denver has about 600,000 inhabitants (2010) and is the capital of the U.S. state of Colorado. We consider a 6×6-grid between *Larimer Street* and *Stout Street* and between *15th Steet* and *20th Street.* Again, the signal settings of the 36 intersections could be extracted from a VISSIM-scenario. All traffic signals are pretimed signals with a cycle time of 75 seconds. The network is presented in Figure 47.

These two scenarios show a typical North-American inner-city grid consisting of regularly arranged one-way streets. For better evaluating the interplay between our traffic assignment and the signal coordination we were interested in a more complex street network. Thus, we chose the German cities of Braunschweig and Cottbus to test our model on a more European shaped city.

6.1.2 Braunschweig and Cottbus

The city of Braunschweig with its 250,000 inhabitants is located in the German federal-state of Lower Saxony and its founding dates back in the 9th century. The presented scenario was proposed by Sándor Fekete and his working group, who also provided most of the data. The *Bohlweg* road leads from the *Technical University of Braunschweig* to the main station. Due to a–most likely–bad coordination of several traffic signals for

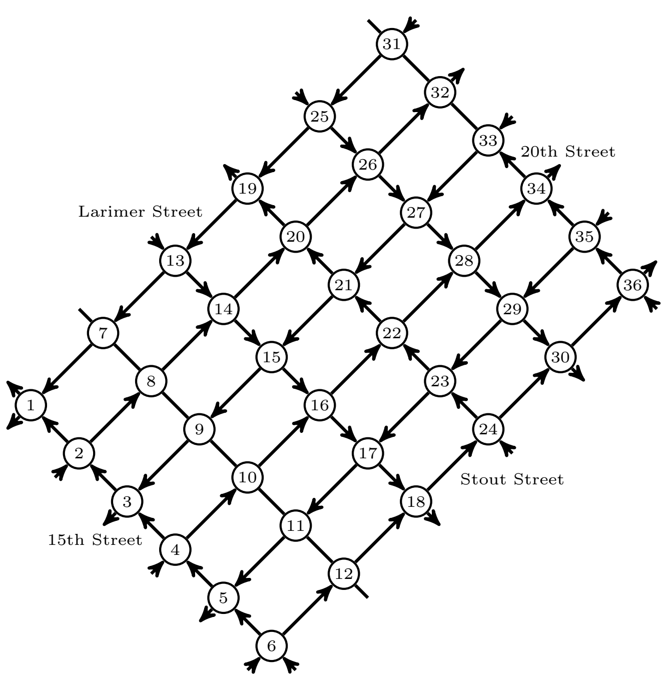

Figure 47: Network of downtown Denver, Colorado. The square between *Larimer Street* and *Stout Street* and between *15th Steet* and *20th Street* consists of 36 intersections with pretimed traffic signals.

pedestrians, the road is completely congested during rush-hour. The network is depicted in Figure 48.

The 10 traffic signals in the Braunschweig scenario have a cycle time of 84 seconds. With 7 signals in a row the network is ideal to test whether our model creates green waves. For simulation, we also created a model of this network for MATSim and VISSIM.

The most complex network in our experiments represents the inner-city of Cottbus (Figure 49). Located in the federal state of Brandenburg and about 75 miles south of Berlin, it is the largest city in the district of Upper and Lower Lusatia with a population of slightly more than 100,000. The settlement was founded by Sorbs in the 10th century on a sandy island in the River Spree. The name 'Chotibuz' was first mentioned in records in 1156.

The scenario consists of the whole inner-city between Bahnhofstraße, Stadtring, and Nordring. The *Brandenburg University of Technology* is located in the Northwest near intersection 19. The historical city-center can be found near intersections 26 – 29. The network was created with data of the OpenStreetMap project. All intersections but one are equipped with actuated traffic signals. The signal settings were measured by the author. Most signals are also influenced by a priority for public transport. Nevertheless, we will use the base plans without changes of phase durations. Commodities were estimated with the traffic observed everyday in mind.

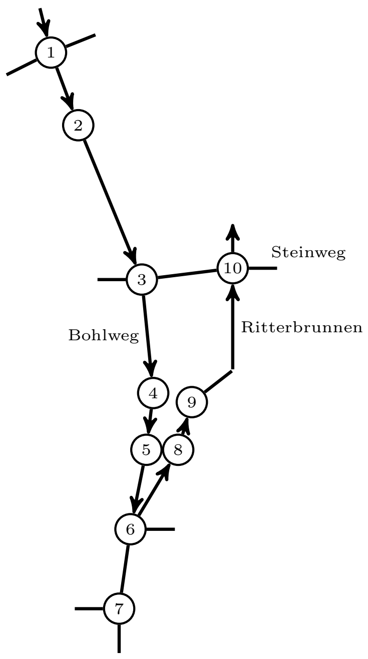

Figure 48: Graph of the inner-city of Braunschweig. The *Bohlweg* leads from to University in the North to the main station in the South. Arcs denote one-way streets. Links with no arrow can be used in both directions. The traffic signals at 2, 4, 5, 8, and 9 enable crossing for pedestrians. They connect tram stations and two shopping malls. Some minor roads are not displayed.

6.2 Test procedure

It takes more than real-world data to evaluate our cyclically time-expanded model. In this section, we specify the remaining test procedure and highlight some essential factors. For each scenario, our modus operandi consists of the following steps.

1. acquisition of data
2. creation of the corresponding cyclically time-expanded model
3. solving of the mixed integer program with CPLEX
4. construction of simulation models in MATSim and VISSIM
5. transfer of assignment and coordination from MIP solution to simulation
6. simulation runs with present coordination, optimized coordination, and random coordinations
7. evaluation of the simulation runs

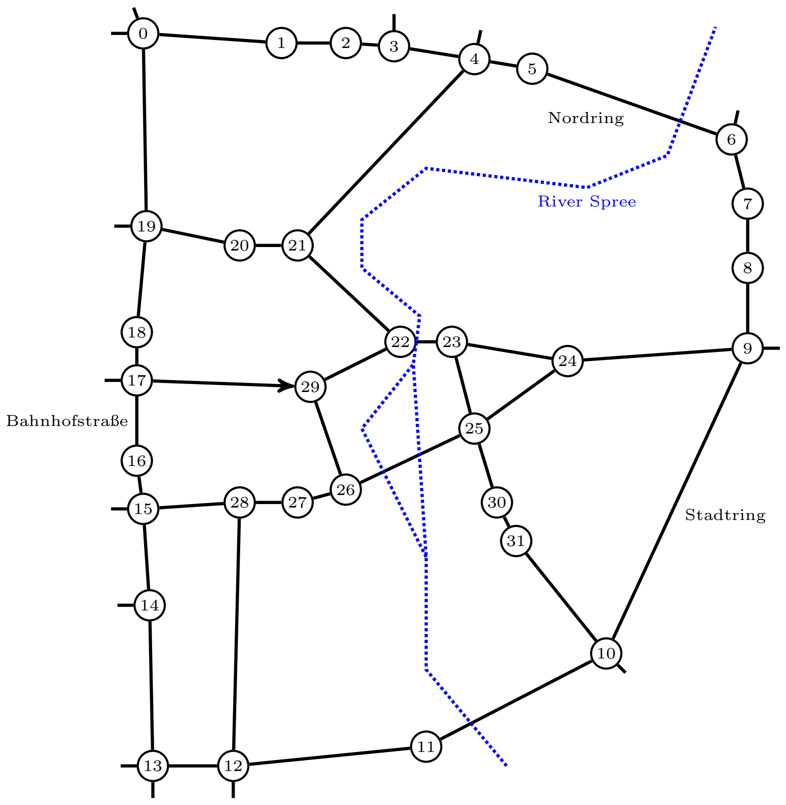

Figure 49: Network of the inner-city of Cottbus with an area of about 7.5 km^2. Only main streets are considered. Most traffic signals are actuated signals. In contrast, traffic signals 8, 11, 30, and 31 are adaptive. Intersection 29 is not equipped with a traffic signal. Meanwhile, intersection 6 is transformed to a roundabout.

6.2.1 Data processing

The scenarios and their generation were already presented in the previous section. However, two important sets of parameters are still missing. We decided to determine transit times and capacities in an easy and documented manner for every scenario. Furthermore, these parameters are not adjusted afterwards to improve the performance of a single instance. Parameter tuning always contains in itself the danger of a harmonization with other coordination or simulation models on a special problem instance instead of providing an independent approach. This concern is also addressed by using two different simulation tools for evaluation. Thus, the transit time of a single road is simply calculated by (geometrical length of the road)/(speed limit on this road). Traffic literature specifies a capacity of 1,800 to 2,000 cars per hour on a single lane. For ease of use, we apply a maximum capacity of one car every two seconds on each lane in our networks. Therefore, the capacity of a road is determined by multiplying the lane capacity of 0.5 cars per second with the number of lanes of a road.

Given a network, the time expansion and the formulation of the mixed integer program is straightforward. We developed our own data format for storing cyclically time-expanded networks with traffic signals. A *C++*-script expands the networks, generates the MIPs and addresses them to CPLEX.

In contrast, transferring the network data and solution to the simulation tools is by no means easy to accomplish. First, the simulation models for MATSim and VISSIM had to be created. Especially VISSIM needs additional geometric information on lanes, traffic signals etc. Thus, the simulation models can hardly be generated from the pure network data automatically and a lot of time consuming manual work was necessary.

Furthermore, the solution of our model has to be transferred to the simulation models. Traffic signals are excellently supported in both simulation tools. With the help of a suitable naming convention, it is easy to copy offset values with a script. However, this is complicated by the fact that we also optimize the assignment. Both simulation tools handle route choice in different ways.

In VISSIM, each commodity has a source, a sink, and a prescribed and fixed route. Hence, we calculate a path decomposition of our assignment. Afterwards, we create new commodities in VISSIM with a one-to-one correspondence to the paths of the path decomposition. The demands are adjusted to the flow values on the paths.

MATSim is based on *agents*. Each agents has a set of *plans* and it executes one of them. Each plan consists of a source, a sink, and a route as well as a departure time. Thus, each agent has a limited route choice. Furthermore, MATSim can generate new plans for agents by shortest path calculations. An executed plan is evaluated and the set of plans of an agent is updated similar to a genetic programming approach. Therefore, we create basic plans for each path in the path decomposition. To transfer the demand, we copy these basic plans in proportion to the flow and adjust the departure times accordingly. Now, each agent is equipped with a unique plan. In the simulation process, replanning is disabled.

6.2.2 Simulation

As mentioned before, for verifying and comparing our solutions to solutions of other optimization techniques we use two different simulation software tools. We already presented MATSim and VISSIM in Section 4.3.5.

Despite their different simulation approaches, both MATSim and VISSIM provide measured travel times for the commodities and they visualize delays and congestion. Additionally, we can also use our cyclically time expanded model to predict travel times and delays, where delay corresponds to the accumulated waiting time at a certain traffic light. Thus, we can compare the solutions of the two simulation tools and our model in terms of these characteristic numbers. However, we have to be careful comparing the optimized solution to the present coordination. Since we also optimize the traffic assignment, delays are of limited information. Instead, we will consider *traffic induced costs*. Assume, each road user travels on the fastest path in an empty network without traffic signals. Summing these free speed transit times for all traffic participants yields a lower bound on the travel time in the network. This trivial bound cannot be underbid and we can only hope to minimize the additional travel time caused by signals and traffic.

The difference to the measured total travel time is called *traffic induced cost* (TIC) and we will measure the improvement of a coordination by means of this term.

Furthermore, the simulation in VISSIM is influenced by stochastical variations, e.g. small changes in speed. Hence, each scenario was simulated at least 20 times and we always present average values in the following. Moreover, each simulation run starts with an empty network. Thus, the travel times of the first cars in a run provide little information. Therefore, we use an initial phase of one hour of simulated time. Afterwards, we measure travel times of all cars in the network for a period of 5 hours of simulated time. Additionally, every scenario was examined with different flow values reaching from nearly empty streets to nose-to-tail traffic by scaling of demands.

Since we do not have coordination data of all of the considered cities or since some signals are influenced by public transport, we also compare the optimized set of offsets to sets of randomly chosen offsets. Although we are aware of the limited informative value of random coordinations, the best random solution gives an impression of what can be obtained by blind guessing. For every scenario, we also randomized 100 sets of random offsets. For each set we used the cyclically time-expanded model to predict congestion. Afterwards, the most promising offsets were simulated with the present assignment as well as the optimized assignment obtained with our model.

6.3 Installing green waves

We start our case studies with the Braunschweig scenario (cf. Figure 48). With several consecutive traffic signals, this scenario admits to test whether the cyclically time-expanded model creates green waves. We consider three commodities. The first commodity (300 cars per hour) starts at intersection 1 and travels to intersection 7. The second commodity with 150 cars per hour origins at *Steinweg* (intersection 10) and joins the first commodity at intersection 3. It also ends at intersection 7. The third commodity (300 cars per hour) travels in opposite direction from intersection 7 to 10. Table 2 and Figure 50 depict the improvement of the present coordination.

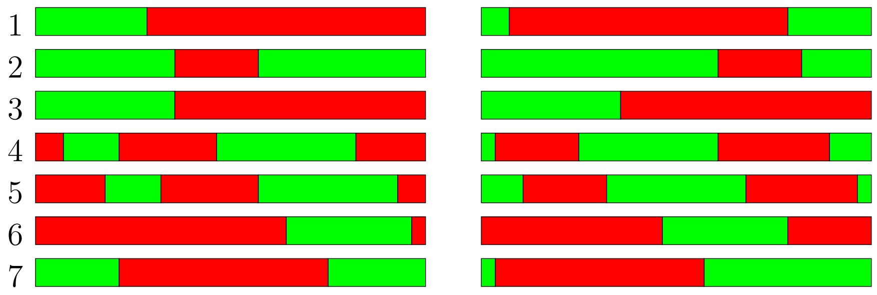

Figure 50: Present (left) and optimzed coordination (right) for the *Bohlweg* in Braunschweig. The signal plans from intersection 1 to intersection 7 are depicted. They are shifted in time with respect to the offsets in Table 2.

intersection	transit time	present	optimized
1		0	67
	10		
2		48	69
	17		
3		0	0
	12		
4		6	76
	7		
5		15	81
	8		
6		54	39
	8		
7		(62)	47

Table 2: Present and optimized offsets for the *Bohlweg* in the Braunschweig scenario. The cycle time is 84 seconds. Furthermore, free speed transit times between consecutive intersections are presented in the second column. Note that the present offset of intersection 7 could not be extracted from the data. Thus, the relative offset of the optimized solution was used in the simulation for both coordinations.

Figure 50 demonstrates that a green wave is created by the cyclically time-expanded model. The present coordination also appears suitable. However, these offsets do not match to the transit times. The differences in performance are clearly visible, when we compare the simulated travel times in Table 3.

commodity	present	optimized	free speed
1	145	98	64
2	86	85	47
3	63	63	44
avg.	100	81	52

Table 3: Average travel times in seconds simulated by VISSIM in the Braunschweig scenario for the three commodities.

The optimized coordination reduces the average traffic induced costs per road user from 47.8 seconds to 28.8 seconds. Thus, about 40 % less delay is realized. Even for higher demands, a significant reduction of traffic induced cost is possible with the same coordination (see Table 4).

The improvement is not only visible in the tables above. The good performance of our coordination is also observable directly in the simulation. Figure 51 shows details of screenshots made in VISSIM.

commodity	present	optimized
1	163	104
2	90	96
3	65	65
avg.	109	87

Table 4: Average travel times in seconds for the Braunschweig scenario with doubled demands. Now, 1500 cars per hour enter the network. The travel time of commodity 1 improves considerably compared to the present coordination. However, cars of commodity 2 need slightly more time.

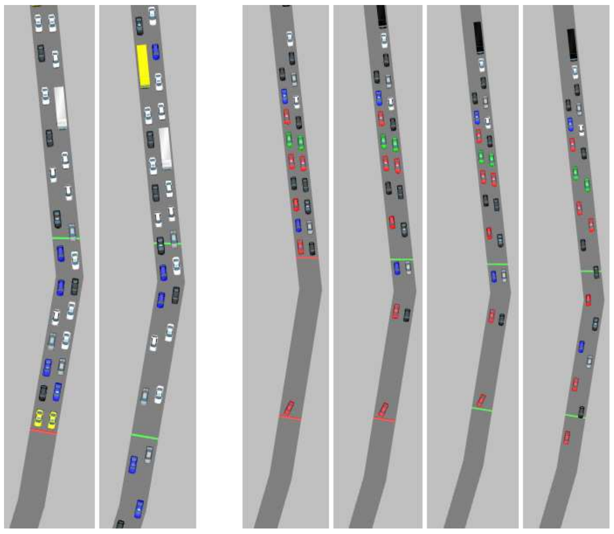

Figure 51: Details of screenshots made in VISSIM (ptv AG). Traffic signals at intersection 4 and 5 are shown. In the present coordination on the left side, a *red* signal blocks the traffic such that cars have to wait in front of a *green* signal at intersection 4. In the coordination optimized with the cyclically time-expanded network and shown on the right side, traffic signal 5 turns *green* just in time when the platoon of cars is arriving.

6.4 Comparison to established approaches

To compare our model to previous approaches we use the results of Wünsch [154] for the Portland scenario and a part of the Denver scenario. Network data and simulation results were kindly provided by the author of [154]. The benchmark is set by TRANSYT (TRAffic Network StudY Tool, version Transyt-7F 10.1) and compared to Wünsch's platoon model (cf. Section 4.3.2).

As a reference, we also compare our solution to the present coordination. Unfortunately, it is not known how this coordination was obtained. Most likely, it was designed by a traffic engineer by hand. Further, we include random offsets in our experiment.

Since all benchmark scenarios use prescribed routes, we disabled route choice in our cyclically time-expanded model, i.e., we compare only the performance of coordination. Due to the one-way streets, route choice has no big influence anyway.

Table 5 summarizes the simulation results for the Portland scenario. The running time of the particular algorithm which calculated the coordination is given in brackets.

coordination	delay
average random	30.1
best random	21.9
present	16.6
CYC. TIME-EXPANSION (10 s)	16.4
PLATOON MODEL (1 s)	16.1
TRANSYT (10 s)	22.2
TRANSYT (800 s)	15.9

Table 5: Simulation results for Portland (VISSIM).

For coordination without the assignment our approach yields competitive solutions. The differences between the solutions most likely result from the differences of the traffic model. TRANSYT uses a model with dynamics similar to the model in the simulation tool VISSIM; the platoon approach and our approach make much more simplifying assumptions. Furthermore, we did not use a fine-tuned calibration of travel times in our model, yet. This will probably yield even less delay.

A similar result is obtained for the Denver scenario. This scenario has already a very good coordination. Still, the solution of our cyclically time-expanded model slightly reduces the average travel time from 117 to 113 seconds. A MATSim screenshot showing delays is depicted in Figure 52. However, the present coordination applies to a larger network. Thus, our coordination omits maybe some constraints of surrounding traffic signals. More seriously, again only little rerouting occurs due the network structure with one-way roads.

6.5 Advantages of simultaneous assignment

One of the main intentions of our model is feedback between traffic assignment and coordination during the optimization process. We demonstrate the impact of our approach by the help of a small, idealized example. Given two parallel routes between two intersections A and B with two additional intersections on each route (Figure 53, we aim to find an optimal assignment for two commodities (A to B and B to A) and an optimal coordination for the six traffic signals. We assume identical operating sequences for each traffic signal (cycle time 60 s, green for 27 s) and a travel time of 20 seconds between consecutive intersections.

For initially computing a static traffic assignment we assume that the same strictly convex link-performance function is given for each link in the network. A conventional approach would first determine the assignment by the help of these functions. Convexity

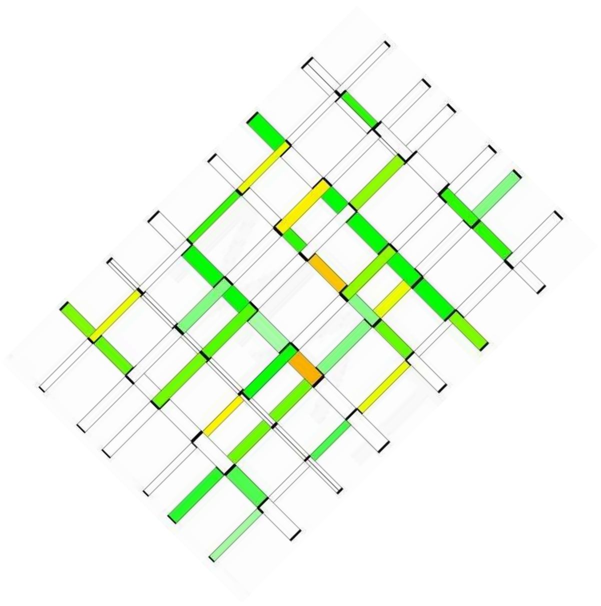

Figure 52: Visualisation of delays in the Denver scenario in MATSim. White links indicate a perfect coordination. On green links a small amount of waiting time is accumulated. Only a few roads show significant congestion (orange).

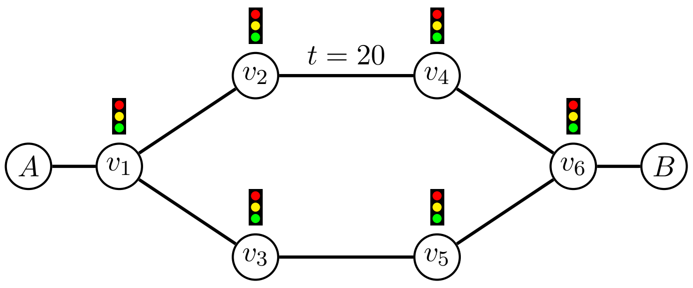

Figure 53: Idealized example for demonstrating the impact of simultaneous routing and offset optimization.

yields a fifty-fifty split for each commodity between upper and lower path in this simple network. Therefore, we fix this split and optimize the coordination with our model and compare this to the results of the simultaneous optimization.

For the fifty-fifty split there are many conflicts of traffic in opposite directions. The simulation yields an average waiting time of 25.2 seconds for the best coordination.

In contrast, the computed solution of the simultaneous optimization with our model is very different. The commodity from A to B is completely assigned to the lower path, while the commodity from B to A is completely assigned to the upper path. This avoids all conflicts with opposite traffic and allows perfect ‘green waves’. The average waiting time is reduced to 6.1 seconds and occurs only at the first traffic signal where cars are assumed to appear randomly. The simultaneous assignment and coordination reduces waiting time by 75 percent in this simple scenario.

Of course, the parameters in the example above were chosen ideally for maximum effect. Furthermore, we did not consider any crossing traffic. However, the intended advantage of simultaneous optimization in contrast to a successive approach is clearly visible even in this very simple example. The same effect of separating traffic flows to force 'green waves' is also observed in realistic scenarios like Cottbus, although the effect on the waiting times is not as dramatic as in the constructed network. To prove this we chose the following approach. First of all, determine a classical, static traffic assignment with VISUM. VISUM is a traffic planning tool from ptv AG linked with VISSIM and features various state-of-the-art static traffic assignment techniques. Secondly, fix this assignment in our model to the route choice proposed by VISUM and optimize the traffic signal coordination. Note that it is not trivial to assign flow to a certain path, because it is not known which time slots will be chosen in the cyclically time-expanded network. Instead, a new commodity is introduced for each path of the path decomposition of the VISUM assignment. For each commodity, all arc capacities except those corresponding to the arcs of the path are set to zero such that the flow is fixed to the route but it can move freely in time. Thirdly, optimize signals and assignment simultaneously for the same demands with our model and compare both solutions.

We chose the Cottbus scenario with 17 origin-destination pairs for this experiment. The chosen demands correspond to 200 cars that enter the network during one cycle of 90 seconds. Optimizing with both strategies we observed that free route choice yields 11 % less additional traffic induced travel time (see Table 6 and Figures 54–57). Furthermore, VISUM assigns most commodities to their shortest paths and therefore opposing commodities to the same road. In contrast, the cyclically time-expanded network model suggests a circulation around the city center. More road users are assigned to the outer circular road, bypassing the center. Nearly two third of them use the outer road in clockwise direction, and only one third uses this road in counterclockwise manner.

	total travel time in seconds	traffic induced costs in seconds	pure waiting time in seconds
fixed routes	36,736	8,967	8,528
free assignment	**35,729**	**7,960**	4,821
best random	40,345	12,576	-
free speed	27,769	-	-

Table 6: The traffic-induced costs are reduced from 9,000 to about 8,000 seconds. These 1,000 seconds are saved every cycle, i.e. every 90 seconds. Remember that the traffic induced costs do not only consist of the overall waiting time. In the case of free assignment they also include additional travel times for detours which are accepted to avoid congested roads. The total waiting time is about 4,800 seconds per cycle.

These completely different but high-performance assignments characterize the presented model. We believe that similar efficient results cannot be obtained without simultaneously considering assignment and coordination, i.e., a highly-adaptive traffic signal optimization will have problems to find a competitive solution. These results also suggest that traffic signal coordination could be used to actively route or redirect traffic.

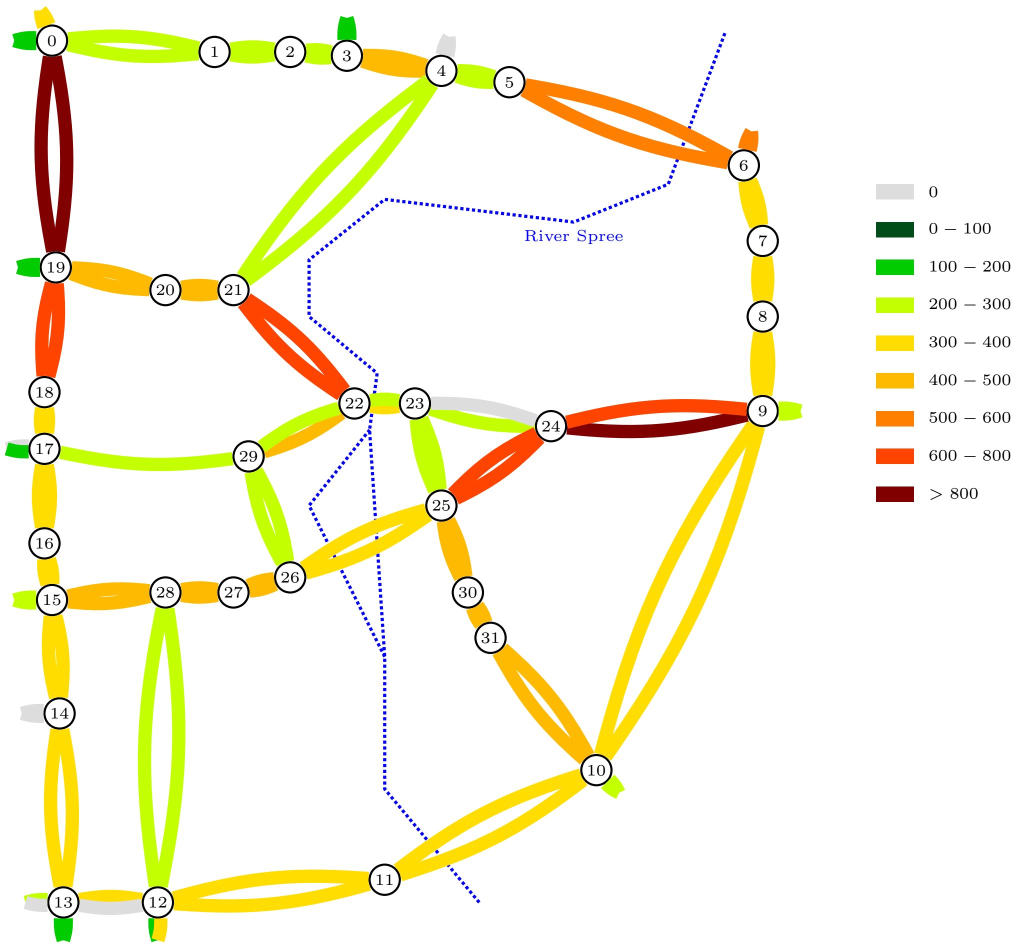

Figure 54: Traffic assignment calculated with VISUM for 17 commodities in the Cottbus scenario. The color encodes the traffic density from dark green (up to 100 cars per hour and lane) to dark red (more than 800 cars per hour and lane) separately for both directions. It is notable that the traffic is distributed fairly equally in the network. Only a few commodities deviate from their shortest paths due to exhausted capacities. Some changes in traffic density are caused by capacity changes, i.e. a varying number of lanes.

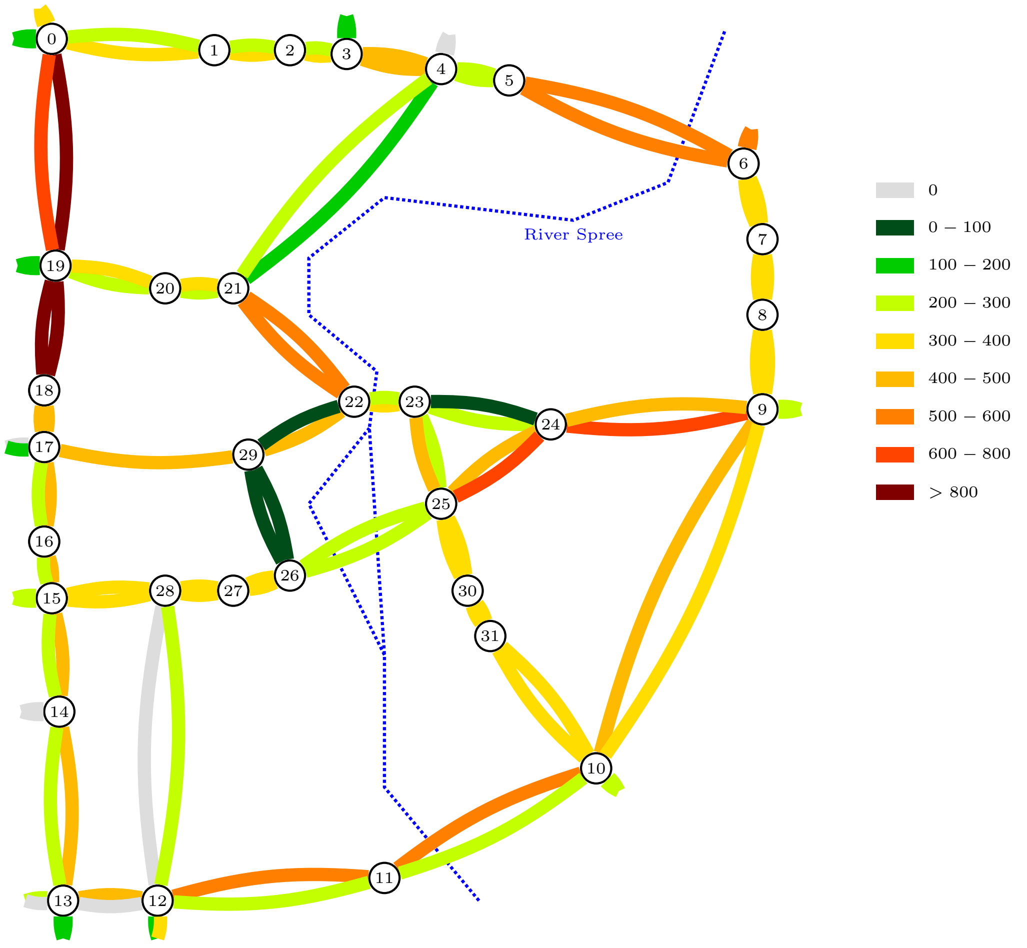

Figure 55: Traffic assignment for the same scenario as in Figure 54, calculated with the cyclically time-expanded model. The traffic on the outer road is increased in clockwise direction.

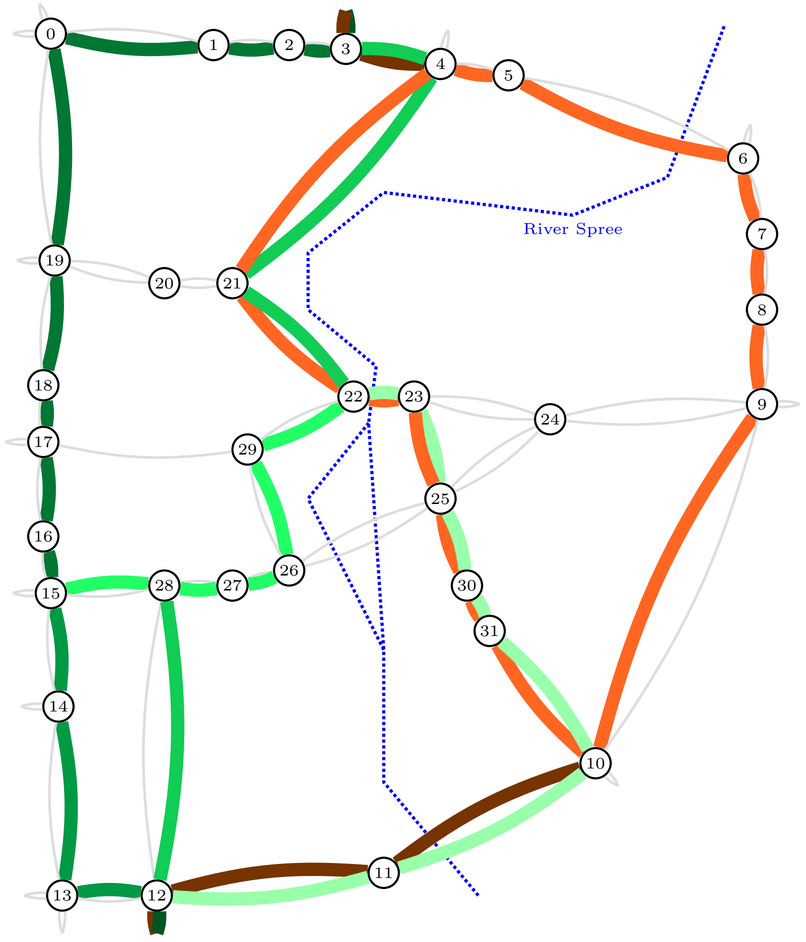

Figure 56: Splitting of two commodities in the scenario of Figure 55. 400 road users per hour travel from a shopping mall in the North (intersection 3) to a housing area in the South (intersection 12). Their paths are displayed in red. The same amount of cars is traveling in the opposite direction (green paths). VISUM assigns all flow units to the shortest path which runs straight through the city center (not shown in the diagram). In our model more than 50% of the road users join the circulation on the outer road. Here, the color brightness encodes the traffic density (dark means higher density).

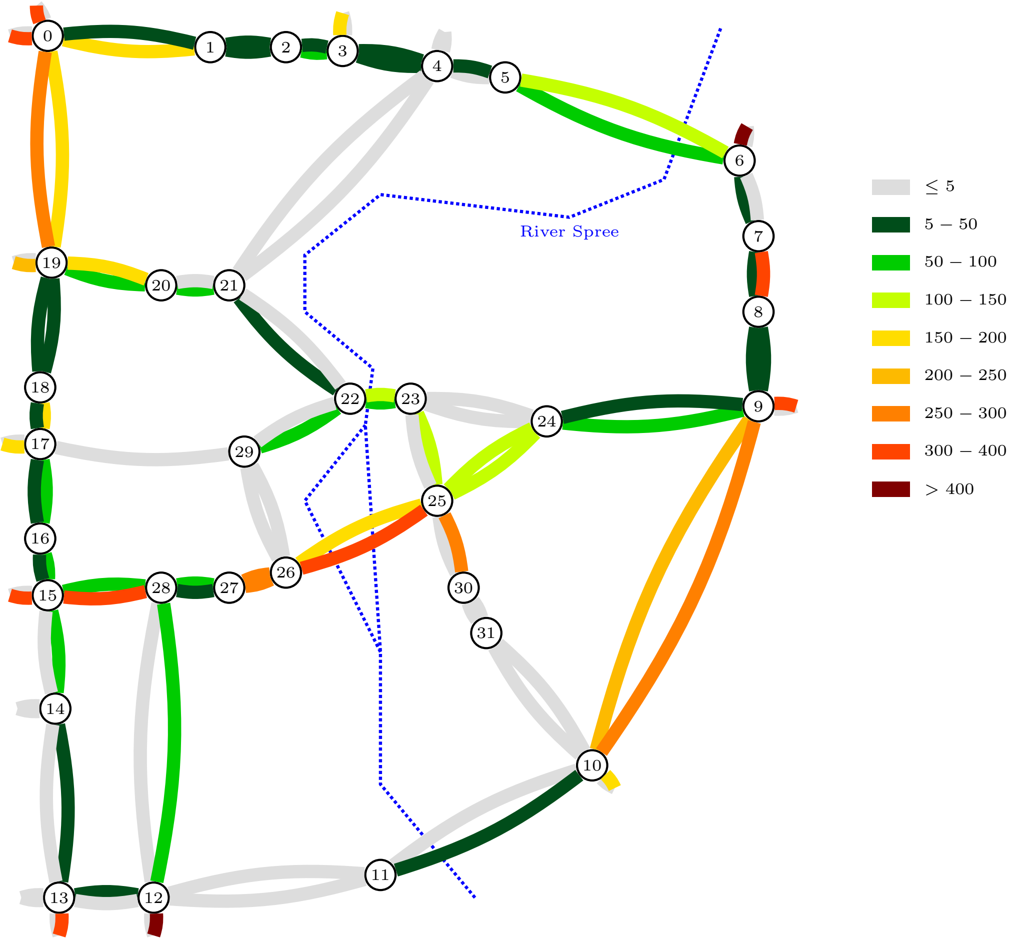

Figure 57: This diagram shows the waiting time integrated over all cars during one cycle at each traffic signal after optimization with our model. Dark green means total waiting time of less then 50 seconds, dark red stands for more than 400 seconds at the specific traffic signal. With the assignment in Figure 55 in mind the waiting time on streets with high traffic volume is rather low. A good coordination is found. The highest waiting times occur at the feeder roads where traffic is equally distributed over time.

6.6 Solving the MIP

For practical applications a model should be quickly solvable. Due to the time-expansion and the decision variables, the MIP-formulation of our model is rather large. Table 7 shows the size of the Cottbus scenario with its 31 traffic signal controlled intersections, 14 commodities, and a cyclic expansion with 90 time steps ($\Gamma = 90$ seconds) and the corresponding mixed integer program (MIP). The offset of one traffic signal is fixed to reduce symmetry.

	Number of	
Network:	nodes	15,570
	arcs	31,410
	decision variables	2,700
MIP:	variables	80,211
	constraints	159,821
	non-zeroes	625,167

Table 7: Size of the expanded Cottbus network and its MIP.

Solving the mixed integer programs with CPLEX produces some expected and some surprising effects. Optimal or near to optimal solutions are obtained very fast. For low or medium traffic load, a good assignment and coordination for the regularly shaped city of Portland is often calculated in less than 10 seconds and the optimal solution is obtained in less than 60 seconds. For high load, the convergence is slower, but good solutions are still computed quite fast. Unfortunately, it takes much longer to actually prove that these solutions are optimal or close to optimal, i.e. to close the gap to the dual bound. The dual bound increases slowly while the primal solution remains unchanged (see also Table 8).

In scenarios with many conflicts between commodities with opposite directions, e.g. Cottbus, and high load the gap cannot be closed at all by CPLEX. In some scenarios a gap of up to 30 % remains even after hours of computation. Still, the obtained primal solutions show very good performance in the simulation. On the other hand, even slight disturbances like changing the demand of a commodity or changing a signal plan a little bit may yield a completely different behavior of the solver and may significantly influence the running time. For example, changing the phase order of a single signal plan can change the computation time by a factor of 10 and parameter tuning in CPLEX does not give a uniform picture. In this section, we will mainly focus on these hard instances.

The highest hurdle is the computation of good dual bounds. This observation is supported by data from the Cottbus scenario. The instance in Table 7 was solved using CPLEX 12. We obtained a primal solution with objective function of value 44,100. The best dual bound stayed at 37,025 even after one day of computation. To rate the relevance of this dual solution we compare it to two trivial bounds. Calculating the length of a shortest path for each commodity times the demand of this commodity yields a dual bound of 33,390. Calculating an assignment without traffic signals in the unexpanded network yields a dual bound of about 35,180. Thus, the dual bound was not significantly increased by CPLEX.

In the following, we present our efforts on the improvement of the formulation of the mixed integer program to speed up the computation of good lower bounds. Unfortunately, we had only little success so far. We will briefly summarize our experiences and the occurring difficulties.

6.6.1 Influence of granularity

Up to now, we did not discuss granularity of our model in detail. From the practical point of view, a granularity of one time step per second seems to be reasonable. With a typical maximum flow rate of one car per 2 seconds and lane, we have to limit the capacity of an arc in our model to 1/2 flow units per time unit. So, thinking of common disturbances in real world traffic, this time expansion is fine enough.

In contrast, a rougher granularity yields fewer binary variables and the size of the MIP is significantly reduced. With rougher granularity, we have to adapt the matrix Q of the interior traffic signal plans, because the provided signal plans have a precision of one second. This can be done by reducing or increasing green times. Therefore, the value of the optimal solution will increase due to worse coordination, but it may also change due to the inaccuracy of the matrix Q, i.e. increased *green* times. We study the influence of granularity in the Portland scenario. We choose a scenario with very high load and slow convergence. The results are presented in Table 8.

Granularity:	$1s$	$3s$	$6s$
initial	21,273/6,477	27,810/6,576	34,632/6,612
after 10 s	11,532/6,479	10,995/6,621	10,692/7,314
after 60 s	10,198/6,479	10,251/7,107	9,888/7,770
after 300 s	9,300/6,479	9,504/7,260	9,888/8,334
after 600 s	9,300/6,601	9,504/7,386	9,888/8,616
after 1 h	9,180/6,610	9,504/7,779	9,888/9,144
after 5 h	8,769/6,725	9,504/8,283	9,888/9,888

Table 8: Solution values (primal/dual) and calculation times for different choices of granularity for the Portland scenario with routing enabled and very high traffic load. For the finest granularity, the primal solution value stayed at 8,713 even after two days of calculation, but the dual solution was slowly increasing to 7,049. Assuming a behavior as for the other granularities, the primal solution after 5 minutes seems to be only 6 % off of the optimum. Hence, convergence is very poor, but an optimal or near to optimal solution is calculated rather fast.

A finer granularity slows down the convergence, but competitive solutions are still obtained quite fast even for the finest cyclic time expansion. The primal solution after

60 seconds of this finest expansion is only 3 % away from the optimal solution of the roughest expansion. After five minutes, the solution with the finest granularity outperforms all solutions for rougher time steps.

On the other hand, it is much harder to compute lower bounds and prove optimality for finer granularity. However, the dual bounds of the rougher cyclic time expansions cannot be used for the finer expansions.

From a practical point of view, the better solutions are obtained for finer expansions. From a theoretical perspective, optimality certificates – or at least good bounds – can be obtained for rougher expansions. From an optimist's point of view, the primal gap seems to be much smaller than the dual gap.

Nevertheless, one may use a successive refinement to find good solutions even faster. Given a solution for a rough granularity, we create the cyclically time-expanded network for a finer granularity, but we admit offsets only near the given solution, i.e. several binary decision variables are set to 0 a priori. With a successive refinement good solutions can be found faster. Table 9 shows computations times for this approach and the Braunschweig scenario. In this case, both solution values are also equal. This is not true for arbitrary scenarios. Since we restrict our model to a subset of possible offsets, we loose the guarantee on the optimality of our solution. Thus, one of our main objectives is not reached. In most applications the decreased running time should not compensate this disadvantage.

Granularity	computation time	
# of steps	original	refined
14	37 s	37 s
28	160 s	2 s
84	2,700 s	8 s
		47 s

Table 9: Computation times for solving the MIP of the successively refined Braunschweig scenario (CPLEX 12). In each refinement step only 10 offsets near the best offset of the previous solution are variable at each traffic signal. An optimal solution for the finest granularity was found in 47 seconds instead of 45 minutes.

6.6.2 Column generation

Column generation often yields significant improvements of running times. We discuss two different approaches.

First, we considered a path based formulation of the traffic assignment problem. This formulation is exponential in size. But with column generation, i.e. increasing the set of active path iteratively, min-cost-flow problems on large networks can often be solved quickly [2].

For our model, we also implemented this approach, but we faced the following problem:

changing one offset may turn the complete set of active paths infeasible. At this moment, one loses all information and the iterative approach fails. One may now think of a slightly relaxed path generation. We do not fix the paths in the cyclically time-expanded network but in the original network. For example, consider a path in the original network with ten traffic signals. With a cyclic time expansion of 60 steps (e.g. a cycle time of 60 seconds and one step per second), this implies a set of 60^{10} paths. Instead of this large set, we fix the corresponding path in the original network. However, the set of reasonable paths in the original inner-city network is small. For most commodities, not more than 10 paths have to be taken into account. Furthermore, even with each commodity fixed to one such path, the problem is still very hard to solve. Thus, no time saving could be achieved.

Second, one may think of a pattern generation approach for the binary offset variables. Pattern generation is very helpful in many problems, e.g. cutting stock, to tackle the combinatorial variety. Therefore, one may interpret the binary decision variables as a pattern. But in our model, exactly one offset per intersection has to be fixed and each combination of offsets for several intersections is feasible in principle. Thus, our patterns have a very easy structure. Even more restricting, we are only allowed to choose exactly one pattern in the solution. Hence, the relation to other pattern generation problems is very poor and we can stop following this approach even without discussing how to generate new patterns out of old ones.

6.6.3 Relaxation and Branch&Cut.

Relaxation and Branch&Cut are two standard techniques for solving mixed integer programs.

First, we consider the relaxation of our model, i.e., the binary conditions for the decision variables b_t^n are dropped and we add the constraint $0 \leq b_t^n \leq 1$ instead. Choosing one offset out of a set of k offsets is still a tight constraint, i.e. $\sum_{t=1}^{k} b_t^n = 1$ itself provides integer feasible solutions. Hence, the difficulties occur in the inequalities for the capacity bounds that are multiplied with the decision variables.

When solving this relaxed formulation one obtains shared offsets. Several of the decision variables range in between 0 and 1. So to say, each traffic light is a little bit *green* and a little bit *red* at the same time. Thus, this relaxation itself has no practical meaning. Not surprising, the solution value is exactly the same value of the first dual bound computed by CPLEX.

Therefore, we have to re-introduce the binary conditions via a Branch&Cut technique. However, we did not succeed in developing a good branching strategy up to now. We tried a blockwise branching, i.e. we shrank the interval of possible offsets in the style of a binary search. For example, the whole set of consecutive offsets from b_1^n to $b_{k/2}^n$ is fixed to zero. Hence, the offset has to be somewhere in the interval $(k/2, k]$. The biggest difficulty is computing good bounds for such a strategy.

The cyclically time-expanded model has a lot of very good solutions that are near to optimal. Some of these solutions differ only slightly, other solutions imply quite different routings and coordinations. Symmetries in the network tighten the problem. This also makes it very difficult to calculate good bounds, i.e. estimations on the traffic induced costs. For each fixed decision, there are various re-routings and changes in the other offset variables such that the optimal value does not change significantly. Hence, the branching tree becomes very large and pruning is hardly possible.

This observation is also supported by the results presented in Figure 58. We took a near to optimal solution from the Cottbus scenario. All but one offset are fixed and we calculate the objective value with respect to the free offset. There occur three local minima for quite different choices of the free offset. Furthermore, Figure 58 suggests that there are various local optima in general. Thus, a local search strategy will also fail.

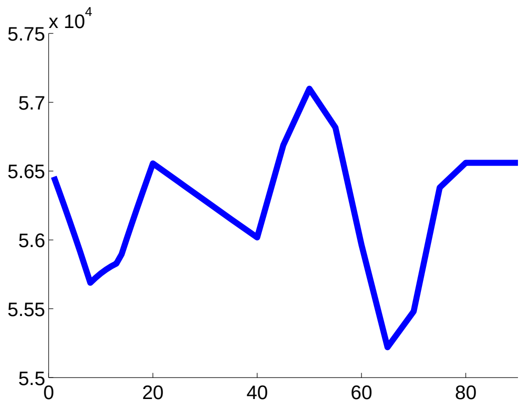

Figure 58: Objective values for the Cottbus scenario. In a random coordination, all but one offsets are fixed. The free offset (Intersection 15) is varied in steps of 5 seconds and optimal assignments are calculated. The corresponding total travel time is displayed in the diagram.

6.7 Conclusion

The cyclically time-expaned model represents a perfect trade-off between static and dynamic models. It provides enough dynamic behavior to capture traffic signals, but it does not carry time dependency to excess.

Another main advantage of the presented cyclically time-expanded model is the simultaneous description of the traffic assignment problem and the traffic signal coordination problem in an uncomplicated linear mixed integer programming formulation. The interactions between road users and the traffic signal coordination are taken into account and provide considerable potential for better solutions. Furthermore, our model allows for an efficient solution of the traffic assignment problem when the signal coordination is fixed.

The simulation results with MATSim and VISSIM suggest the applicability of our model in real-world traffic optimization frameworks. In spite of the size of the MIP-formulation good solutions are obtained quickly. In our solutions green waves, i.e., progressive signal systems, are indeed created and traffic is assigned to well coordinated arterial roads. The optimized coordination features lower travel times and less congestion. The traffic induced costs are usually reduced by 10 to 75 % compared to conventional two-step approaches for assignment and coordination.

Furthermore, our model assumptions of flow independent transit times are not too simplifying. The link performance predicted by the cyclically time-expanded model can be reproduced with the microscopic traffic simulation VISSIM. Quite surprisingly, both simulation tools VISSIM and MATSim, despite their different approaches, produce very similar results. Average travel times vary less than 5 % and the overall picture of congestion is nearly identical. As a consequence, the coordinations obtained by our optimization seem to be robust with respect to disturbances as both simulations rate them equally; in particular, our optimization is not fine-tuned to fit the special properties of a special simulation tool.

Summarizing, our model yields competitive solutions in a competitive time. The advantage of a simultaneous optimization of traffic assignment and traffic signal coordination is clearly visible. Solving this mixed integer program with a common solver also yields a dual bound for the solution. This guaranteed maximal gap to optimality allows to evaluate the quality of the found solution and, thus, this is an advantage over heuristic approaches like genetic programming or neural networks. Most state-of-the-art traffic signal optimization tools like TRANSYT are very vulnerable to local optima and do not provide any dual bound or any guarantee on the optimal solution at all. To the best of our knowledge, there are no other models for a simultaneous optimization of traffic assignment and traffic signal coordination, that use strict mathematical programming and that do not drop significant properties of inner-city traffic.

Nevertheless, there remain some open tasks. The extension of the model to split optimization as presented in Section 5.6.1 is very fascinating from the author's point of view. This will involve mainly two questions. Much more data will be needed to model a single intersection. All safety constraints have to be formulated and integrated into the mixed integer program. Building a scenario like Cottbus will be even more time consuming. Further, the number of decision variables will increase significantly. It will be much harder to compute good solutions, but this approach also provides great potential for better traffic signal coordinations.

With overload, the cyclically time-expanded model simply turns infeasible. Especially in a dynamic setting, one would expect that overload could be handled by extending the time horizon, but the cyclic expansion provides no possibility to expand the time horizon. Thus, other strategies for handling overload or scaling demands are needed. We have also discussed the difficulty of determining stable user equilibria in Section 5.5.3. A concept of equilibria that consider 'pushing' even without time as an indicator for causality, is an interesting challenge. In the present model, road users of a commodity could intentionally block other traffic participants to reduce traffic flow on their remaining path which would reduce travel times for this commodity. Thus, user equilibria involve many aspects and they are also linked to the handling of overload.

Combining both topics of this thesis also yields an interesting question. Most likely, there will always be some road users that are unfamiliar with the network. These users will follow guideposts to their destination. A small percentage of the traffic can be routed explicitly on a fixed route by a proper installation of these traffic signs. The influence of some commodities routed confluently on the network capacity could be much greater. This reminds of *Stackelberg routing* in game-theoretic approaches for traffic assignment. Furthermore, one may study whether traffic signal coordination can be used to affect assignments systematically. Urban planners may be interested in calming traffic near schools or hospitals. 'Red waves' are certainly not the best idea, but providing well-coordinated alternative routes is a reasonable possibility.

Expanding the model directly leads to another open question. The differences in the performance of the solver even for very similar scenarios are not well understood, yet. A good idea for a branch&cut strategy with a clever estimation of traffic induced costs will directly lead to decreased computation times. Additionally, one may include other solvers in the experiments.

The investigations of our cyclically time-expanded model can be extended to other criteria like robustness. What happens with the system if one signal loses contact to the main server and therefore its cycle. Furthermore, more accurate information and data for other cities would enhance our portfolio of scenarios and improve the practical value of our solutions.

Eventually, let us return to the question from the beginning of this thesis. What can mathematics do to support the quest for stress-free, environment-friendly, and safe traveling? For example, guideposts and traffic signals can be optimized with the help of network flow theory to reduce congestion. We presented two new concepts concerning confluent flows and offset optimization. Now, improving guideposts in a real application or testing one of our optimized coordinations in reality would cap it all.

Bibliography

[1] J. Adolph. *Regelung des Wagenverkehrs in Straßen mittels elektrischer Signallampen.* Reichspatentamt, 1925. Patent DE 439255 A.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, 1993.

[3] M. Aigner and G. M. Ziegler. *Proofs from THE BOOK.* Springer, Berlin, 1998.

[4] R. E. Allsop. Selection of offsets to minimize delay to traffic in a network controlled by fixed-time signals. *Transportation Science*, pages 1–13, 1968.

[5] R. E. Allsop and J. A. Charlesworth. Traffic in a signal-controlled road network: An example of different signal timings inducing different routeings. *Traffic Eng. Control*, 18(5):262–265, 1977.

[6] E. Almasri and B. Friedrich. Online offset optimisation in urban networks based on cell transmission model. In *Proceedings of the 5th European Congress on Intelligent Transport Systems and Services (ITS)*, Hannover, Germany, 2005.

[7] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Matrix Analysis and Applications*, 8(2):277–284, 1987.

[8] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *Journal Algorithms*, 12:308–340, 1991.

[9] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation. Combinatorial optimization problems and their approximability properties.* Springer, Berlin, 1999.

[10] G. Baier. *Flows with Path Restrictions.* PhD thesis, TU Berlin, 2003.

[11] G. Baier, E. Köhler, and M. Skutella. On the k-splittable flow problem. In R. Möhring et al., editors, *Proceedings of the 10th annual European Symposium on Algorithms*, number 2461 in LNCS, pages 101–113. Springer, 2002.

[12] G. Baier, E. Köhler, and M. Skutella. The k-splittable flow problem. *Algorithmica*, 42(3-4):231–248, 2005.

[13] M. Becker and K. Mehlhorn. Algorithms for routing in planar graphs. *Acta Informatica*, 23:163–176, 1986.

[14] M. Beckmann, C. McGuire, and C. Winston. *Studies in the Economics of Transportation.* Yale University Press, New Haven, 1956.

[15] M. C. Bell and R. D. Bretherton. Ageing of fixed-time traffic signal plans. In *2nd International Conference on Road Traffic Control*, London, 1986.

[16] M. G. H. Bell and H. Ceylan. Traffic signal timing optimization based on genetic algorithm approach, including drivers' routing. *Transportation Research Part B*, 38:329–342, 2004.

[17] A. Bley. *Routing and Capacity Optimization for IP Networks.* PhD thesis, TU Berlin, 2007.

[18] H. L. Bodlaender. Dynamic programming on graphs of bounded treewidth. In *Proceedings of the 15th International Colloquium on Automata, Languages and Programming*, LNCS 317, pages 105–118, Berlin, 1988. Springer.

[19] H. L. Bodlaender. A Tourist Guide through Treewidth. *Acta Cybernetica*, 11:1–23, 1993.

[20] H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[21] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *22nd International Symposium on Mathematical Foundations of Computer Science*, pages 29–36. Springer, 1997.

[22] P. Bose. On embedding an outer-planar graph in a point set. *Computational Geometry*, 23(3):303–312, 2002.

[23] D. Braess. Über ein Paradoxon aus der Verkehrsplanung. *Unternehmensforschung*, 12:258–268, 1968.

[24] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Special Graph Classes — A Survey.* SIAM Monographs on Discete Mathematics and Applications. SIAM, 1999.

[25] Bundesministerium für Verkehr, Bau- und Wohnungswesen, editor. *Verkehr in Zahlen 2003/2004.* Deutscher Verkehrs-Verlag, Hamburg, 2003.

[26] R. Burkard, K. Dlaska, and B. Klinz. The quickest flow problem. *Zeitschrift für Operations Research*, 37:31–58, 1993.

[27] G. Chartrand and F. Harary. Planar permutation graphs. *Ann. Inst. H. Poincare (section B)*, 3(4):433–438, 1967.

[28] C. Chekuri, S. Khanna, and B. F. Shepherd. A note on multiflows and treewidth. *Algorithmica*, 54:400–412, May 2009.

[29] J. Chen, R. D. Kleinberg, L. Lovász, R. Rajaraman, R. Sundaram, and A. Vetta. (Almost) tight bounds and existence theorems for single-commodity confluent flows. *Journal of the ACM*, 54(4), 2007.

[30] J. Chen, R. Rajaraman, and R. Sundaram. Meet and merge: approximation algorithms for confluent flows. *Journal of Computer and System Sciences*, 72(3):468–489, 2006.

[31] S.-W. Chiou. *Optimization of Area Traffic Control for Equilibrium Network Flows*. PhD thesis, University College London, 1999.

[32] S.-W. Chiou. Joint optimization for area traffic control and network flow. *Computers and Operations Research*, 32:2821–2841, 2005.

[33] S.-W. Chiou. Optimization for signal setting problems using non-smooth techniques. *Information Sciences*, pages 2985–2996, 2009.

[34] Y.-C. Chiu, J. Bottom, M. Mahut, A. Paz, R. Balakrishna, T. Waller, and J. Hicks. A primer for dynamic traffic assignment. Technical report, ADB30 Transportation Network Modeling Committee of the Transportation Research Board, 2010.

[35] S. Cook. The complexity of theorem proving procedures. In *Proceedings of the third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[36] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley, New York, 1998.

[37] J. R. Correa, A. S. Schulz, and N. E. Stier-Moses. Selfish routing in capacitated networks. *Mathematics of Operations Research*, 29(4):961–976, 2004.

[38] B. Courcelle and M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comp. Sc.*, 109:49–82, 1993.

[39] C. F. Daganzo. The cell transmission model: A dynamic representation of highway traffic consistent with the hydrodynamic theory. *Transportation Research B*, 28(4):269–287, 1994.

[40] C. F. Daganzo. The cell transmission model. part ii: Network traffic. *Transportation Research B*, 29(2):79–93, 1995.

[41] C. F. Daganzo. The nature of freeway gridlock and how to prevent it. In J. Lesort, editor, *Traffic and Transportation Theory*, pages 629–646. Pergamon-Elsevier, 1996.

[42] C. F. Daganzo. In traffic flow, cellular automata = kinematic waves. *Transportation Research B*, 40(5):396–403, 2006.

[43] C. F. Daganzo. Urban gridlock: macroscopic modeling and mitigation approaches. *Transportation Research part B*, 41:49–62, 2007.

[44] G. B. Dantzig. *Linear Programming and Extensions*. RAND Corporation, Santa Monica, CA, 1963.

[45] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research 2*, pages 393–410, 1954.

[46] J. de Dios Ortúzar and L. G. Willumsen. *Modelling Transport*. Wiley, Chichester, 2006.

[47] Deutsche Presse-Agentur. News report June 11th, 2009.

[48] T. J. Dickson. A note on traffic assignment and signal timings in a signal-controlled road network. *Transportation Research B*, 15B(4):267–271, 1981.

[49] R. Diestel. *Graph Theory*. Springer, Heidelberg, 4th edition, 2010.

[50] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[51] P. Donovan, B. Shepherd, A. Vetta, and G. Wilfong. Degree-constrained network flows. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, STOC '07, pages 681–688, New York, NY, USA, 2007. ACM.

[52] D. Dressler and M. Strehler. Polynomial-time algorithms for special cases of the maximum confluent flow problem. Submitted.

[53] D. Dressler and M. Strehler. Capacitated confluent flows: Complexity and algorithms. In *Proceedings of the 7th International Conference on Algorithms and Complexity (CIAC 2010)*, 2010.

[54] T. Erlebach. Approximation algorithms for edge-disjoint paths and unsplittable flow. In E. Bampis et al., editors, *Efficient Approximation and Online Algorithms*, LNCS 3484, pages 97–134. Springer, 2006.

[55] T. Erlebach and K. Jansen. The maximum edge-disjoint paths problem in bidirected trees. *SIAM Journal on Discrete Mathematics*, 14(3):326–355, 2001.

[56] Eurostat. `http://epp.eurostat.ec.europa.eu/portal/page/portal/transport/data/database`. Last accessed July 28th, 2011.

[57] F. Fischer and C. Helmberg. Dynamic graph generation and dynamic rolling horizon techniques in large scale train timetabling. In T. Erlebach and M. Lübbecke, editors, *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, volume 14 of *OpenAccess Series in Informatics (OASIcs)*, pages 45–60, Dagstuhl, Germany, 2010.

[58] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[59] L. R. Ford and D. R. Fulkerson. *Flow in Networks*. Princeton University Press, Princeton, 1962.

[60] Forschungsgesellschaft für Straßen- und Verkehrswesen, Köln. *Handbuch für die Bemessung von Straßenverkehrsanlagen*, 2009.

[61] Forschungsgesellschaft für Straßen- und Verkehrswesen, Köln. *Richtlinien für Lichtsignalanlagen*, 2010.

[62] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10:111–121, 1980.

[63] B. Friedrich. Adaptive signal control: an overview. In *Proceedings of the 9th Meeting of the Euro Working Group Transportation*, Bari, 2002.

[64] B. Friedrich. Verkehrsadaptive Steuerung von Lichtsignalanlagen – ein Überblick. Technical report, Institut für Verkehrswirtschaft, Straßenwesen und Städtebau, Universität Hannover, 2002.

[65] A. Fügenschuh, H. Homfeld, and H. Schuelldorf. Routing cars in rail freight service. In *Dagstuhl Seminar Proceedings*, 09261, Dagstuhl, Germany, 2009.

[66] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, 1979.

[67] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. Research Report MPI-I-97-1-025, Max-Planck-Institut für Informatik, Saarbrücken, Germany, November 1997.

[68] N. H. Gartner, J. D. C. Little, and H. Gabbay. Optimization of traffic signal settings by mixed-integer linear programming, Part I: The network coordination problem. *Transportation Sciences*, 9:321–343, 1966.

[69] N. H. Gartner, C. J. Messer, and A. K. Rathi, editors. *Revised Monograph on Traffic Flow Theory: A State-of-the-Art Report.* Turner-Fairbank Highway Research Center, Federal Highway Administration, U.S. Dept. of Transportation, 2005. Available at `http://www.tfhrc.gov/its/tft/tft.htm`.

[70] R. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society 64*, pages 275–278, 1958.

[71] N. Goyal, N. Olver, and F. B. Shepherd. The VPN conjecture is true. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 443–450, New York, NY, USA, 2008. ACM.

[72] P. Gritzmann, B. Mohar, J. Pach, and R. Pollack. Embedding a planar triangulation with vertices at specified points (solution to problem E3341). *The American Mathematical Monthly*, 98(2):165–166, 1991.

[73] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1:169–197, 1981.

[74] T. Hagerup, J. Katajainen, N. Nishimura, and P. Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57:366–375, December 1998.

[75] A. Hall, S. Hippler, and M. Skutella. Multicommodity flows over time: Efficient algorithms and complexity. *Theor. Comput. Sci.*, 379(3):387–404, 2007.

[76] R. W. Hall, editor. *Handbook of Transportation Science.* Kluwer Academic Publishers, Boston, Dordrecht, London, 2nd edition, 2003.

[77] M. Heckhausen. *Das Buch vom Ampelmännchen.* Eulenspiegel, 1997.

[78] B. Hoppe. *Efficient dynamic network flow algorithms.* PhD thesis, Cornell University, 1995.

[79] B. Hoppe and É. Tardos. The quickest transshipment problem. *Mathematics of Operations Research*, 25:36–62, 2000.

[80] P. Hunt, D. Robertson, R. Bretherton, and R. Winton. Scoot – a traffic responsive method of co-ordinating signals. Technical report, Traffic Reasearch Laboratory, 1981. Laboratory Report 1014.

[81] International Maritime Organization. *Convention on the International Regulations for Preventing Collisions at Sea*, 1972.

[82] T. Johnson, N. Robertson, P. D. Seymour, and R. Thomas. Directed tree-width. *Journal of Combinatorial Theory, Series B*, 82(1):138–154, 2001.

[83] V. Kaibel and M. A. F. Peinhardt. On the bottleneck shortest path problem. Technical Report 06-22, Konrad-Zuse-Zentrum f. Informationstechnik Berlin, 2006.

[84] N. K. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[85] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, New York, 1972.

[86] M. Kaufmann and G. Klär. A faster algorithm for edge-disjoint paths in planar graphs. In W.-L. Hsu and R. Lee, editors, *ISA'91 Algorithms*, volume 557 of *Lecture Notes in Computer Science*, pages 336–348. Springer Berlin / Heidelberg, 1991.

[87] L. G. Khachiyan. A polynomial algorithm in linear programming. *Sov. Math., Dokl.*, 20:191–194, 1979.

[88] J. Kleinberg. *Approximation algorithms for disjoint paths problems.* PhD thesis, MIT, 1996.

[89] J. M. Kleinberg. Single-source unsplittable flow. In *37th Annual Symposium on Foundations of Computer Science*, pages 68–77, 1996.

[90] B. Klinz and G. Woeginger. Minimum-cost dynamic flows: The series-parallel case. *Networks*, 43:153–162, 2004.

[91] B. Klinz and G. J. Woeginger. One, two, three, many, or: Complexity aspects of dynamic network flows with dedicated arcs. *Operation Research Letters*, 22:119–127, 1998.

[92] R. Koch, M. Skutella, and I. Spenke. Maximum k-splittable s,t-flows. *Theory of Computing Systems*, 43:56–66, 2008.

[93] E. Köhler, K. Langkau, and M. Skutella. Time-expanded graphs with flow-dependent transit times. In R. H. Möhring and R. Raman, editors, *Proceedings ESA 2002*, LNCS 2461, pages 599–611. Springer, 2002.

[94] E. Köhler, R. H. Möhring, and M. Skutella. Traffic networks and flows over time. In J. Lerner, D. Wagner, and K. A. Zweig, editors, *Algorithmics of Large and Complex Networks*, LNCS 5515, pages 166–196. Springer, 2009.

[95] E. Köhler, R. H. Möhring, and G. Wünsch. Minimizing total delay in fixed-time controlled traffic networks. In *Proceedings of Operations Research (OR)*, 2004.

[96] E. Köhler and M. Strehler. Traffic signal optimization using cyclically expanded networks. Journal submission.

[97] E. Köhler and M. Strehler. Traffic signal optimization using cyclically expanded networks. In T. Erlebach and M. Lübbecke, editors, *Proceedings of the 10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems*, OpenAccess Series in Informatics (OASIcs), 2010.

[98] E. Köhler and M. Strehler. Signalanlagenoptimierung mit zyklisch expandierten Netzwerken. In *Proceedings of HEUREKA 11*, pages 314–333, Köln, 2011. FGSV Verlag.

[99] G. Kolata. What if they closed 42d street and nobody noticed? *New York Times*, December 25th 1990.

[100] B. Korte and J. Vygen. *Combinatorial Optimization – Theory and Algorithms.* Springer, Berlin, 4th edition, 2007.

[101] M. R. Kramer and J. van Leeuwen. Wire-routing is NP-complete. Technical report, Rijksuniversiteit Utrecht, 1982.

[102] S. Lämmer. *Reglerentwurf zur dezentralen Online-Steuerung von Lichtsignalanlagen in Straßennetzwerken.* PhD thesis, Technische Universität Dresden, 2007.

[103] S. Lämmer. Stabilitätsprobleme vollverkehrsabhängiger Lichtsignalsteuerungen. Technical report, Technische Universität Dresden, 2009. Diskussionsbeitrag.

[104] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[105] W. Li. Power spectra of regular languages and cellular automata. *Complex Systems*, 1:107–130, 1987.

[106] J. D. C. Little. The synchronizing of traffic signals by mixed-integer linear programming. *Operations Research 14*, pages 568–594, 1966.

[107] J. Lynch. The equivalence of theorem proving and the interconnection problem. *SIGDA Newsletter*, 5(3):31–36, 1975.

[108] S. Mamada, K. Makino, and S. Fujishige. Optimal sink location problem for dynamic flows in a tree network. *IEICE Trans. Fundamentals*, E85-A:1020–1025, 2002.

[109] S. Mamada, K. Makino, and S. Fujishige. An evacuation problem in tree dynamic networks with multiple exits. In *Proceedings of International Symposium on Systems & Human Science for Safety, Security, and Dependability*, pages 347–351, 2003.

[110] S. Mamada, T. Uno, K. Makino, and S. Fujishige. A tree partitioning problem arising from an evacuation problem in tree dynamic networks. *Journal of the Operations Research Society of Japan*, 48(3):196–206, 2005.

[111] S. Mamada, T. Uno, K. Makino, and S. Fujishige. An $O(n \log^2 n)$ algorithm for the optimal sink location problem in dynamic tree networks. *Discrete Appl. Math.*, 154(16):2387–2401, 2006.

[112] MATSim Homepage. `http://www.matsim.org/`. Last accessed October 17th, 2011.

[113] D. Meuren. Die rot-grüne Koalition. *Der Spiegel*, 2001. Available online `http://www.spiegel.de/kultur/gesellschaft/0,1518,159072,00.html`.

[114] S. L. Mitchell. Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Information Processing Letters*, 9:229–232, 1979.

[115] R. H. Möhring, K. Nökel, and G. Wünsch. A model and fast optimization method for signal coordination in a network. In H. van Zuylen and F. Middelham, editors, *Proceedings of the 11th IFAC Symposium on Control in Transportation Systems - CTS 2006*, pages 73–78, Delft, The Netherlands, 2006.

[116] J. T. Morgan and J. D. C. Little. Synchronizing traffic signals for maximal bandwidth. *Journal of the Operations Research Society of America*, 12(6):896–912, 1964.

[117] K. Nagel and G. Flötteröd. Agent-based traffic assignment: Going from trips to behavioral travelers. In *Proceedings of the 12th Conference of the International Association for Travel Behaviour Research (IATBR)*, Jaipur, India, 2009.

[118] K. Nagel and D. Grether. Personal communication, 2010.

[119] K. Nagel and M. Schreckenberg. A cellular automaton model for freeway traffic. *J. Phys. I France 2*, pages 2221–2229, 1992.

[120] J. Nash. Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295, 1951.

[121] R. Niedermeier. *Invitation to fixed-parameter algorithms.* Oxford University Press, Oxford, 2006.

[122] J. Nocedal and S. J. Wright. *Numerical optimization.* Springer, New York, 2nd edition, 2006.

[123] H. Okamura and P. D. Seymour. Multicommodity flows in planar graphs. *Journal of Combinatorial Theory*, 31:75–81, 1981.

[124] M. Papageorgiou, C. Diakaki, V. Dinopoulou, A. Kotsialos, and Y. Wang. Review of road traffic control strategies. In *Proceedings of the IEEE*, volume 91 (12), pages 2043–2067, 2003.

[125] T. Pfender. Arboreszenz-Flüsse in Graphen: polyedrische Untersuchungen. Diploma thesis, Technische Universität Berlin, 2001.

[126] ptv AG. *VISSIM manual.* Last accessed September 6th, 2011.

[127] B. Reed, N. Robertson, A. Schrijver, and P. Seymour. Finding disjoint trees in planar graphs in linear time. In N. Robertson et al., editors, *Graph structure theory. Proceedings of the AMS-IMS-SIAM joint summer research conference on graph minors*, pages 295–301. American Mathematical Society, 1993.

[128] D. I. Robertson. TRANSYT method for area traffic control. *Traffic Engineering & Control*, 10:276 – 281, 1969.

[129] N. Robertson and P. Seymour. Graph minors. xiii: The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[130] R. P. Roess and W. R. McShane. *Traffic Engineering.* Prentice Hall, Englewood Cliff, New Jersey, 1999.

[131] T. Roughgarden. The price of anarchy is independent of the network topology. *Journal of Computer and System Sciences*, 67:341–364, September 2003.

[132] T. Roughgarden. *Selfish Routing and the Price of Anarchy.* MIT Press, Cambridge, 2005.

[133] T. Roughgarden and É. Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002.

[134] P. Scheffler. A practical linear time algorithm for disjoint paths in graphs with bounded treewidth. Technical Report 396, Technische Universität Berlin, 1994.

[135] Schlothauer & Wauer Ingenieurgesellschaft für Straßenverkehr. *Sylvia+ – Kurzbeschreibung*, 2010.

[136] D. Schrank and T. Lomax. 2009 Urban Mobility Report. Texas Transportation Institut, 2009. Available at `http://tti.tamu.edu/documents/mobility_report_2009_wappx.pdf`.

[137] A. Schrijver. Finding k disjoint paths in a directed planar graph. *SIAM J. Comput.*, 23(4):780–788, 1994.

[138] A. Schrijver. *Combinatorial Optimization – Polyhedra and Efficiency*. Springer, Berlin, 2003.

[139] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM Journal on Discrete Mathematics*, 2(4):550–581, 1989.

[140] P. Serafini and W. Ukovich. A mathematical model for the fixed-time traffic control problem. *European Journal of Operational Research*, 42(2):152–165, 1989.

[141] Y. Sheffi. *Urban Transportation Networks: Equilibrium Analysis with Mathematical Programming Methods*. Prentice Hall, 1984.

[142] M. Skutella. An introduction to network flows over time. In W. Cook, L. Lovász, and J. Vygen, editors, *Research Trends in Combinatorial Optimization*, pages 451–482. Springer Berlin Heidelberg, 2009.

[143] D. Sun, R. F. Benekohal, and S. T. Waller. Multiobjective traffic signal timing optimizing using non-dominated sorting genetic algorithm. In *IEEE IV2003 Intelligent Vehicles Symposium*, 2003.

[144] W. Y. Szeto and H. K. Lo. Dynamic traffic assignment: Properties and extensions. *Transportmetrica*, 2(1):31–52, 2006.

[145] F. Teklu, A. Sumalee, and D. Watling. A genetic algorithm approach for optimizing traffic control signals considering routing. *Computer-Aided Civil and Infrastructure Engineering*, 22:31–43, 2007.

[146] Transportation Research Board, Washington, D.C. *Highway Capacity Manual*, 4th edition, 2000.

[147] R. J. Vanderbei. *Linear Programming*. Springer, New York, 3rd edition, 2008.

[148] D. Wagner and K. Weihe. A linear-time algorithm for edge-disjoint paths in planar graphs. *Combinatorica*, 15(1):135–150, 1995.

[149] A. Warberg, J. Larsen, and R. M. Jørgensen. Green Wave Traffic Optimization – A Survey, 2008.

[150] J. G. Wardrop. Some theoretical aspects of road traffic research. In *Institute of Civil Engineers, PART II*, volume 1, pages 325–378, 1952.

[151] K. Weihe. Multicommodity flows in even, planar networks. In K. Ng, P. Raghavan, N. Balasubramanian, and F. Chin, editors, *Algorithms and Computation*, volume 762 of *Lecture Notes in Computer Science*, pages 333–342. Springer Berlin / Heidelberg, 1993.

[152] D. B. West. *Introduction to Graph Theory.* Prentice Hall, Upper Saddle River, 2nd edition, 2001.

[153] L. A. Wolsey. *Integer programming.* Wiley, Chichester, 1998.

[154] G. Wünsch. *Coordination of Traffic Signals in Networks.* PhD thesis, Technische Universität Berlin, 2008.

List of Figures

Index