Technische Universität Ilmenau 24. Februar 2024

Vorlesung

Netzalgorithmen

Prof. Dr.-Ing Günter Schäfer

NeXTeX

Vorlesungsmitschrift von
Adrian Schollmeyer

Inhaltsverzeichnis

1 Einführung **4**

2 Modeling Network Design Problems **5**
2.1 Modeling Design Problems in Link-Path Formulation 5
2.2 Modeling Design Problems in Node-Link Formulation 6
2.3 Link-Demand-Path-Identifier-Based Notation 6
2.4 Shortest-Path Routing 8

3 General Optimization Methods for Network Design **9**
3.1 Extreme Points and Basic Solutions 9
3.1.1 Phase 2 9
3.2 The Simplex-Algorithm 9
3.2.1 Phase 1 9
3.2.2 Simplex Tableaus 10
3.3 Duality – Motivation 12
3.4 Duality 14
3.5 Duality (7) 14
3.6 Duality (8) 15
3.7 Duality (10) 15
3.8 Branch and Bound 15
3.9 Cutting Planes for MIP 16

4 Network Design Problems **17**
4.1 Simple Design Problem 17
4.2 Capacitated Problems 17
4.3 Path Diversity 18
4.4 Lower Bounded Flows 18
4.5 Limited Demand Split 18
4.6 Modular Flow Allocation 18
4.7 Modular Links 19
4.8 Convex Cost and Delay Functions 19
4.9 Concave Link Dimensioning Functions 20
4.10 Budget Constraints 20
4.11 Incremental Network Design Problems 20
4.12 Representing Nodes 20

5 Network Resilience **22**
5.1 Depth First Search 22

5.2 Verifying Descendant Relationships via Preordering 22
5.3 Low Value of Nodes 22

Stichwortverzeichnis **24**

1 Einführung

2 Modeling Network Design Problems

2.1 Modeling Design Problems in Link-Path Formulation

Preliminary notation for undirected demands:

- $x - y$ ...Link between node x and y
- $\hat{h}_{xy} = b$ ...Demand d between nodes x and y (i. e. Demand b bandwidth between nodes x and y)
- $\hat{x}_{v_1 v_2} = f$ ...Flow on the path between v_1 and v_2
- $\hat{x}_{v_1 v_2 v_3} = f$ ...Flow on the path v_1, v_2, v_3
- $\hat{c}_{v_1 v_2}$ ...Capacity of the link between v_1 and v_2

This generally yields equations for satisfying the demands and inequalities for limiting the bandwidth to the link capacities, e. g.

$$\hat{x}_{12} + \hat{x}_{132} = 5 \tag{2.1}$$
$$\hat{x}_{13} + \hat{x}_{123} = 7 \tag{2.2}$$
$$\vdots \tag{2.3}$$
$$\hat{x}_{132} + \hat{x}_{13} + \hat{x}_{213} \leq 10 \tag{2.4}$$
$$\hat{x}_{132} + \hat{x}_{123} + \hat{x}_{23} \leq 15 \tag{2.5}$$

As such a system usually has multiple solutions, an **objective function** is defined, which can then be optimized, e. g.:

$$F = \hat{x}_{12} + 2\hat{x}_{132} + \hat{x}_{13} + 2\hat{x}_{123} + \hat{x}_{23} + 2\hat{x}_{213} \tag{2.6}$$

In this example, flow routed over two links are weighted with factor two, so direct links are preferred.

Such an optimization is called a **multi-commodity flow problem**. The *optimal solution* to this problem is marked with an asterisk, e. g.:

$$\hat{x}^*_{12} = 5 \qquad \hat{x}^*_{13} = 7 \qquad \hat{x}^*_{23} = 8 \tag{2.7}$$

Important observations:

- Changing the objective function usually affects the optimal solution to a problem.
- Formulation a good objective function for the particular network is important for obtaining meaningful solutions.

2.2 Modeling Design Problems in Node-Link Formulation

In this section, links and demands are assumed to be directed. Undirected links $x - y$ are replaced with two directed links $x \to y$ and $y \to x$.

Notation:

- $v_1 \to v_2$ …Directed link from node v_1 to node v_2
- $\langle v_1 : v_2 \rangle$ …Demand from node v_1 to node v_2
- $\tilde{x}_{a,d}$ …Flow over arc a for demand d (e. g. $\tilde{x}_{v_1 v_2, v_1 v_2}$ flow over arc $v_1 \to v_2$ for demand $\langle v_1 : v_2 \rangle$)

Backflows (e. g. $\tilde{x}_{21,12}$) are also possible, but make practically no sense, so they are set to 0.

2.3 Link-Demand-Path-Identifier-Based Notation

Demands and links are assigned label indexes. Thus, tables for mapping indexes to actual demands and links are required.

Notation:

- h_i …Demand with index i
- c_e …(Known) capacity of link e
- y_e …*Unknown* capacity of link e
- P_i …Number of candidate paths for demand i
- P_{ij} …jth candidate path for demand i
- $(v_1, e_1, v_2, e_2, \ldots, e_n, v_{n+1})$ …n-hop path
- $v_1 \to v_2 \to \ldots \to v_{n+1}$ …node representation of a path (directed, use $-$ instead of $\to$ for undirected)
- $\{e_1, e_2, \ldots, e_n\}$ …link representation of undirected paths
- $(e_1, e_2, \ldots, e_n)$ …link representation of directed paths
- v …Node
- e …Link
- d …Demand
- p …Path
- V, E, D, P …total numbers of the aforementioned items

- ξ_e ...Cost of alink e

Example equations:

$$x_{11} = 15 \tag{2.8}$$
$$x_{21} + x_{22} = 20 \tag{2.9}$$
$$x_{31} + x_{32} = 10 \tag{2.10}$$

Demands:

$$\sum_{p=1}^{P_d} x_{dp} = h_d \qquad d = 1, \ldots, D \tag{2.11}$$

Short form, when iterating over all candidate paths:

$$\sum_{p} x_{dp} = h_d \qquad d = 1, \ldots, D \tag{2.12}$$

The vector of all flows (path flow variables) is called the **flow allocation vector** or short **flow vector**:

$$\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_D) \tag{2.13}$$
$$= (x_{11}, x_{12}, \ldots, x_{1P_1}, \ldots, x_{D1}, x_{D2}, \ldots, x_{DP_D}) \tag{2.14}$$
$$= (x_{dp} : d = 1, 2, \ldots, D; p = 1, 2, \ldots, P_d) \tag{2.15}$$

Important: vectors are represented with bold letters $\mathbf{x}$ and scalar values are represented with normal letters x.

The relationship between links and paths is written down with the **link-path incidence relation** δ_{edp}:

$$\delta_{edp} = \begin{cases} 1 & \text{if link } e \text{ belongs to path } p \text{ for demand } d \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

δ_{edp} can be written as a table:

e	$P_{11} = \{2, 4\}$	$P_{21} = \{5\}$	$P_{22} = \{3, 4\}$
1	0	0	0
2	1	0	0
3	0	0	1
4	1	0	1
5	0	1	0

The **load** in link e can be written as:

$$\underline{y}_e = \underline{y}_e(\mathbf{x}) = \sum_{d=1}^{D} \sum_{p=1}^{P_d} \delta_{edp} x_{dp} \tag{2.17}$$

Important: The actual link loads $\underline{y}_e$ are determined by the path flow variables x_{dp} of a solution and are not the same as the link capacity variables y_e.

Cost of a path P_{dp}:

$$\zeta_{dp} = \sum_e \delta_{edp}\xi_e, \qquad d = 1, 2, \ldots, D \quad p = 1, 2, \ldots, P_d \tag{2.18}$$

Shortest-Path Allocation Rule for Dimensioning Problems For each demand, allocate its entire demand to its shortest path with respect to link costs and candidate paths. If there is more than one shortest path for a given demand, then the demand volume can be arbitrarily split amon the shortest paths.

2.4 Shortest-Path Routing

For shorted-path routing, demands will only be routed on their shortest paths. The path length is determined by adding up link costs w_e according to some weight system $\mathbf{w} = (w_1, w_2, \ldots, w_E)$.

Single Shortest Path Allocation Problem For given link capacities $\mathbf{c}$ and demand volumes $\mathbf{h}$, find a link weight system $\mathbf{w}$ such that the resulting shortest paths are unique and the resulting flow allocation vector is feasible.

Zusammenfassung fortführen

3 General Optimization Methods for Network Design

3.1 Extreme Points and Basic Solutions

Eine Lösung heißt Basislösung, wenn es so viele linear unabhängige Spalten gibt, wie die Matrix insgesamt an Zeilen hat.

3.1.1 Phase 2

$$S(x) \text{ linear unabhängig, wenn } \sum_{i=1}^{k} \alpha_i \vec{x}_i = 0 \implies \forall i \in \{1, \ldots, k\} : \alpha_i = \vec{o} \tag{3.1}$$

3.2 The Simplex-Algorithm

Note: Solving the system of equations means transforming the target vector into the basis of the input vector.

If all d_i are non-negative, the basic solution ist the optimal solution as the $d_i y_i$ are always subtracted and y_i is always non-negative, thus positive d_i can only decrease the result and not increase it.

$\vec{z}_k$ is non-negative for all three cases as $t_{k,j}$ is non-positive and $\alpha > 0$. For items that are not j and not part of the base, the result is 0 and thus it can be removed from the sum.

If we find that all d_j are negative, the objective function is unbounded from above.

In case 3, ε is always positive due to the way it is defined and the constrains on the values in this case. As ε is the minimum of the given set, it can not be < 0.

Thus, the resulting vector $\vec{z}$ can be in the feasible set.

Basically, we have walked from one corner to the next and restarted the algorithm. In each calculation, we display $\vec{b}$ in another basis. Thus, we need to watch out for cycling between multiple results. This is what Bland's rule is used for.

3.2.1 Phase 1

For distinction, a new variable vector $\vec{y}$ is introduced. This is later transformed by extending $\vec{x}$. All y_i are weighted 1.

This auxiliary problem can be solved by setting $\vec{x} = \vec{o}$.

Note that the auxiliary problem is to minimize, not maximize the result. As all y_i need to remain positive, the algorithm will approach $\forall i : y_i = 0$ eventually.

3.2.2 Simplex Tableaus

Page 35 last line: $\vec{\tilde{b}}$ is actually $\vec{b}$...

Example 1:

$$\max x_1 + x_2 \tag{3.2}$$

such that:

$$-x_1 + x_2 \leq 1 \tag{3.3}$$
$$x_1 \leq 3 \tag{3.4}$$
$$x_2 \leq 2 \tag{3.5}$$
$$x_1, x_2 \geq 0 \tag{3.6}$$

This needs to be transformed from canonical form into the standard form. For this, additional variables $x_3, x_4, \ldots$ need to be introduced:

$$-x_1 + x_2 + x_3 = 1 \tag{3.7}$$
$$x_1 + x_4 = 3 \tag{3.8}$$
$$x_2 + x_5 = 2 \tag{3.9}$$
$$\vec{x} \geq \vec{o} \tag{3.10}$$

A basic solution can be obtained directly:

$$x_1 = 0 \tag{3.11}$$
$$x_2 = 0 \tag{3.12}$$
$$x_3 = 1 \tag{3.13}$$
$$x_4 = 3 \tag{3.14}$$
$$x_5 = 2 \tag{3.15}$$

From this, the current maximum value is $z = 0$.

We now work with the simplex tableau.

$$\begin{array}{cccccc} x_1 & x_2 & x_3 & x_4 & x_5 & b \\ -1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 0 & 1 & 2 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{array} \tag{3.16}$$

The last row represents the objective function and the last cell if the last row is z.

This matrix can now be transformed using Gauss-Jordan elimination. During elimination, we need to ensure that $\vec{b}$ remains positive!

We apply Blant's rule and find the first column where the value in the last row is non-zero. Then, we try to set this cell to 0 using the Gauss-Jordan transformation. Additionally, we transform the matrix to get a unity matrix in the first 3 columns. We continue until we have a unity matrix in the first 3 columns (or can transform the system of equations into such a form by changing the order of rows).

Every time a value in the last row is non-zero (except for $-z$, of course), the result can be improved further!

$$\left|\begin{array}{cccccc|l} -1 & 1 & 1 & 0 & 0 & 1 & \leftarrow + \\ 1 & 0 & 0 & 1 & 0 & 3 & \cdot 1 \\ 0 & 1 & 0 & 0 & 1 & 2 & \\ 1 & 1 & 0 & 0 & 0 & 0 & \leftarrow + \end{array}\right. \quad \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right| \begin{array}{l} \\ \\ \cdot -1 \\ + \end{array} \tag{3.17}$$

$$\Longrightarrow \left|\begin{array}{cccccc|l} 0 & 1 & 1 & 1 & 0 & 4 & \leftarrow + \\ 1 & 0 & 0 & 1 & 0 & 3 & \\ 0 & 1 & 0 & 0 & 1 & 2 & \cdot -1 \\ 0 & 1 & 0 & -1 & 0 & -3 & \leftarrow + \end{array}\right. \quad \left.\begin{array}{l} \\ \\ \\ \\ \end{array}\right| \begin{array}{l} \\ \\ \cdot -1 \\ + \end{array} \tag{3.18}$$

Row 1 can not be subtracted from row 3 as it would result in a negative b_3. We can, however, subtract row 3 from row 1. We obtain a matrix where all coefficients in the last row are negative. Thus, we have obtained an optimal solution and $-z$ is in the last cell of the matrix.

$$\Longrightarrow \left|\begin{array}{cccccc} 0 & 0 & 1 & 1 & -1 & 2 \\ 1 & 0 & 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & -1 & -1 & -5 \end{array}\right| \tag{3.19}$$

Thus, we obtain:

$$\vec{x} = \begin{pmatrix} 3 \\ 2 \\ 2 \\ 0 \\ 0 \end{pmatrix} \tag{3.20}$$

This solution indeed satisfies all constraints. After the Gauss-Jordan transformation, we have obtained the values for $t_{k,j}$. From the structure of $\hat{A}$ we see that

$$\left(\hat{A}_N\right)_{k,j} = t_{k,j} \tag{3.21}$$

for all $k \in B$ and $j \in N$. What happens with the $t_{k,j}$ is basically solving three systems of equations at once.

Generic form of the matrix:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \vdots & a_{1n} & b_1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} & b_m \\ c_1 & c_2 & c_3 & \dots & c_n & z \end{pmatrix} \tag{3.22}$$

Example 2:

$$\max x_2 \tag{3.23}$$

such that

$$x_1 - x_2 \leq 1 \tag{3.24}$$
$$-x_1 + x_2 \leq 2 \tag{3.25}$$
$$x_1, x_2 \geq 0 \tag{3.26}$$

First of all, we write this in matrix form. We also introduce slack variables:

$$\left|\begin{array}{ccccc} 1 & -1 & 1 & 0 & 1 \\ -1 & 1 & 0 & 1 & 2 \\ 1 & 0 & 0 & 0 & 0 \end{array}\right| \tag{3.27}$$

$$\Longrightarrow \left|\begin{array}{ccccc} 1 & \mathbf{-1} & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 3 \\ 0 & \mathbf{1} & -1 & 0 & -1 \end{array}\right| \tag{3.28}$$

We can see in the second column that c_2 is positive and all other values negative or 0. This means that the solution is unbounded! Thus, the solution is infinitely high.

Example 3 contains errors!

3.3 Duality – Motivation

Consider the LOP:

$$\max f(\vec{x}) = 2x_1 + 3x_2 \tag{3.29}$$

such that

$$4x_1 + 3x_1 \leq 12 \qquad \text{(I)} \tag{3.30}$$
$$2x_1 + x_2 \leq 3 \qquad \text{(II)} \tag{3.31}$$
$$3x_1 + 2x_2 \leq 4 \qquad \text{(III)} \tag{3.32}$$
$$x_1, x_2 \geq 0 \tag{3.33}$$

While this system of equation has more constraints than variables, we get more variables than constraints as soon as we add slack variables.

As $x_1, x_2 \geq 0$, we have:

$$2x_1 + 3x_1 \leq 4x_1 + 8x_2 \leq 12 \tag{3.34}$$

Thus, 12 is an upper bound for $f(\vec{x})$. If we divide the first equality by 2, we obtain an even better bound

$$2x_1 + 3x_2 \leq 2x_1 + 4x_2 \leq 6. \tag{3.35}$$

Even better, if we calculate $\frac{1}{3} \cdot (\mathrm{I}) \cdot (\mathrm{II})$.

$$\begin{aligned} 2x_1 + 3x_2 &\leq \frac{1}{3}(4x_1 + 8x_2 + 2x_1 + x_2) && (3.36)\\ &\leq \frac{1}{3}(12 + 3) && (3.37)\\ &= 5 && (3.38) \end{aligned}$$

Hence, $f(\vec{x})$ can not get larger than 5. How good can an upper bound get this way? So, we are trying to derive an inequality of the form

$$d_1 x_1 + d_2 x_2 \leq h \tag{3.39}$$

with $d_1 \geq 2$, $d_2 \geq 3$ and h as small as possible. So $\forall x_1, x_2 \geq 0$ we have

$$2x_1 + 3x_2 \leq d_1 x_1 + d_2 x_2 \leq h. \tag{3.40}$$

Combining the 3 inequalities of the original LOP with some non-negative coefficients y_1, y_2, y_3 (so that direction of inequalities is not reversed), we obtain

$$\underbrace{(4y_1 + 2y_2 + 3y_3)}_{d_1} x_1 + \underbrace{(8y_1 + y_2 + 2y_3)}_{d_2} x_2 \leq \underbrace{12y_1 + 3y_2 + 4y_3}_{h} \tag{3.41}$$

How to choose y_i? We need to ensure that $d_1 \geq 2$ and $d_2 \geq 3$ and we want to have h as small as possible under these constraints. So we have a new LOP:

$$\min 12y_1 + 3y_2 + 4y_3 \tag{3.42}$$

such that

$$\begin{aligned} 4y_1 + 2y_2 + 3y_3 &\geq 2 && (3.43)\\ 8y_1 + y_2 + 2y_3 &\geq 3 && (3.44)\\ y_1, y_2, y_3 &\geq 0 && (3.45) \end{aligned}$$

This is called the **dual LOP** to the original LOP.

We now show that *every* feasible solution we can find for the dual LOP is an upper bound for the original LOP and a feasible solution to the original LOP is a lower bound for the dual LOP. The original problem is referred to as the **primal problem**.

$$\max \vec{c}^\top \vec{x} \text{ s. t. } A\vec{x} \leq \vec{b}, \vec{x} \geq \vec{o}(P) \tag{3.46}$$

$$\min \vec{b}^\top \vec{y} \text{ s. t. } A^\top \vec{y} \geq \vec{c}, \vec{y} \geq \vec{o}(D) \tag{3.47}$$

We can see that $\vec{y}$ in the dual problem is multiplied with the *transposed* matrix. Thus, if the primal problems has 2 variables and 3 constraints, the dual problem has 3 variables, but only 2 constraints.

We later show that the optimal solution for both problems is the same and any feasible solution is a bound for the optimal solution. If the primal optimization problem is unbounded, the dual problem has no solution. Vice versa, if the primal problem has no solution, the dual problem is unbounded.

3.4 Duality

Note that we now calculate $y^\top A$ in the slide. Let's recall:

Let A be an $m \times n$ matrix, $\vec{x}$ be an $m \times 1$ vector, $\vec{b}$ be an $m \times 1$ vector.

$$A \cdot \vec{x} = \vec{b} \tag{3.48}$$

$$\iff (A \cdot x)^\top = \vec{b}^\top \tag{3.49}$$

$$\iff x^\top \cdot A^\top = \vec{b}^\top \tag{3.50}$$

Thus, we can remove the parentheses for the transposition by applying the transposition for all inner parts of the parentheses.

We can multiply:

- a row vector and a matrix: $(1 \times m) \cdot (m, n) = (1 \times n)$
- a matrix and a column vector: $(m \times n) \cdot (n \times 1) = (m \times 1)$

3.5 Duality (7)

$(\hat{P})$ expands upon (P) by adding slack variables:

$$\max \vec{c}^\top \vec{x} \tag{3.51}$$

s. t.

$$A\vec{x} + \vec{z} = \vec{b} \tag{3.52}$$

$$\iff \tag{3.53}$$

$$\max \left(\vec{x}^\top, \vec{o}^\top\right) \begin{pmatrix} \vec{x} \\ \vec{z} \end{pmatrix} \tag{3.54}$$

s. t.

$$(A|I)\underbrace{\begin{pmatrix}\vec{x}\\ \vec{z}\end{pmatrix}}_{\vec{\tilde{x}}} = \vec{b} \tag{3.55}$$

Note that $\hat{h}_B$ and $\hat{h}_N$ have nothing to do with h. $\vec{\hat{y}}$ has nothing to do with the $\vec{y}$ from the duality explanation, but it is rather the $\vec{y}$ that was the solution of the LOP as in the simplex algorithm.

3.6 Duality (8)

$$\hat{A}_B^{-1}\hat{A}_B\vec{\hat{x}}_B = \hat{A}_B^{-1}\hat{A}_B\vec{\hat{y}}_B + \hat{A}_B^{-1}\hat{A}_n\vec{\hat{y}}_N \tag{3.56}$$

$$\iff \vec{\hat{x}}_B = \vec{y}_B + \hat{A}_B^{-1}\hat{A}_N\vec{\hat{y}}_N \tag{3.57}$$

$$\iff \vec{\hat{y}}_B = \vec{\hat{x}}_B - \hat{A}_B^{-1}\hat{A}_N\vec{\hat{y}}_N \tag{3.58}$$

As $\vec{\hat{x}}$ is a basic solution, $\vec{\hat{x}}_N = \vec{o}$, so there is no $\vec{\hat{x}}_N$ in the equations.

3.7 Duality (10)

$$\vec{u}^\top A \geq \vec{c}^\top \tag{3.59}$$

$$\vec{u} \geq \vec{o} \tag{3.60}$$

$$\vec{u}^\top := \vec{c}^\top \hat{A}_B^{-1} \tag{3.61}$$

$$\vec{\hat{c}}^\top \hat{A}_B^{-1} A \geq \vec{c}^\top \tag{3.62}$$

$$\vec{\hat{c}}^\top \hat{A}_B^{-1} \geq \vec{o}^\top \tag{3.63}$$

$$A\vec{x} = \vec{b} \tag{3.64}$$

$$\iff \hat{A}_B\vec{\hat{x}}_B = \vec{\hat{b}} \tag{3.65}$$

$$\iff \hat{A}_B^{-1}\hat{A}_B\vec{\hat{x}}_B = \hat{A}_B^{-1}\vec{\hat{b}} \tag{3.66}$$

3.8 Branch and Bound

Achtung: hier wird jetzt minimiert statt maximiert!

- Grundidee: ein schweres Problem mit einem relaxierteren Problem zusammenführen

- $g(x)$ in der relaxierten Version kann etwas verändert formuliert werden, muss aber $= f(x)$ auf R sein.
- Die Lösungen, die uns nicht gefallen, werden schrittweise ausgeschlossen und dann neu gerechnet, um schrittweise der richtigen Lösung näher zu kommen.
- Branch and Bound:
  - allgemeine Algorithmenidee, die darauf beruht, dass eine Menge in zwei Teilmengen partitioniert und zwei Optimierungsprobleme gelöst werden.
  - Von den Optima der Teilmengen, muss dann lediglich der kleinere der beiden gewählt werden
  - Man suche sich zunächst eine optimale Lösung für das relaxierte Problem. Die Lösung erfüllt dann ggf. nicht die Ganzzahligkeitsbedingung. Teile dann das Problem auf in zwei Teilprobleme, wobei der nicht ganzzahlige Parameter einmal kleiner und einmal größer als der optimale Wert des relaxierten Problems ist.
  - Bounding: wenn in einem bestimmten Teilbereich die optimale Lösung größer als das bisher bekannte Optimum ist, rechne dort gar nicht erst weiter, da man dort keine optimale Lösung finden wird.
- Bei der Branch-Operation kommen zusätzliche Nebenbedingungen hinzu, die weitere Schlupfvariablen einführen und damit die Berechnung erschweren. Dafür wird bei diesen Teilproblemen jeweils der Lösungsraum immer kleiner.

3.9 Cutting Planes for MIP

Idee bei Branch and Bound war, immer Teile wegzuschneiden. Idee von Cutting Planes ist jetzt, einen chirurgischen Schnitt zu erzeugen, der eine Ecke so wegschneidet, dass kein Punkt aus M weggeschnitten wird. Schritt für Schritt wird dann der Lösungraum immer weiter kleingeschnitten, bis die Ecken (und damit auch die optimale Lösung) ganzzahlig sind.

Großer Vorteil ist hier, dass sich die Anzahl der zu lösenden Probleme nicht ständig verdoppelt.

4 Network Design Problems

4.1 Simple Design Problem

Der Optimierer kann grundsätzlich aus $\leq y_e$ immer ein $= y_e$ machen, da die Kosten ξ_e immer 1 werden (ansonsten hätte man geringere Kosten). Damit ist es möglich, E viele Slack-Variablen im Optimierungsproblem zu vermeiden und entsprechend die Matrix klein zu halten. Dadurch lässt sich $\sum_d \sum_p \delta_{edp} x_{dp}$ statt y_e einsetzen. Am Ende fallen dadurch nur noch D Nebenbedingungen an.

Durch das ganze Optimierungskram kommt man zu dem Punkt, dass die Lösung eine Shortest-Path-Lösung sein wird, wo nur ein Pfad bei rausfällt. Man kann also genau so gut problemspezifischere Lösungsalgorithmen wie Dijkstra benutzen.

4.2 Capacitated Problems

Das *Pure Allocation Problem* arbeitet ohne Zielfunktion. Kapazitätsmangel kann durch Einführung einer Variablen z berechnet werden, die in den Nebenbedingungen zu jeder Kapazität addiert und dann als Zielfunktion minimiert wird. Dabei kann es auch sein, dass auf Kanten z draufaddiert wird, die genug Kapazität haben. Wenn jetzt z nach Optimierung negativ ist, weiß man, dass genug Kapazität vorhanden ist. Umgekehrt zeigt positives z an, dass nicht genug Kapazität vorhanden ist.

Wenn es eine gültige Lösung gibt, gibt es höchstens $D + E$ Flows, die ≥ 0 sind. Das folgt wieder daraus, dass man höchstens $D + E$ (Anzahl Zeilen) viele Variablen auf $\neq 0$ setzen kann. Wenn die $s_e > 0$, gibt es wieder wegen Single-Path-Allocation genau D viele Flows. Alternativ lässt sich auch die insgesamt ungenutzte Kapazität maximieren. Dabei kann man mit r_e noch angeben, wie wichtig einem ist, dass bestimmte Links nicht ausgelastet werden. Schlimmstenfalls kann man hier auf Gleichheit kommen (d. h. der Link ist ausgelastet). Hier können bspw. Situationen entstehen, bei denen kleine Links mit wenig Priorität voll ausgelastet werden. Minimiert man stattdessen die Rate r, mit der jeder Link ausgelastet wird, kann man eine solche Überlastung einzelner Links besser vermeiden.

Bei der Optimierung sollte man die Konsequenzen der Linkauslastung beachten. Gerade stark oder vollständig ausgelastete Links können nicht gut auf Lastspitzen reagieren, wodurch es zu vermehrtem Paketverlust kommen kann.

4.3 Path Diversity

Bei der Aufteilung von Flüssen muss immer darauf geachtet werden, dass niemals einzelne Flows aufgeteilt werden. Bei der Verteilung einens Flusses auf viele Pfade kann also der maximale Anteil einens Flows zu jedem Fluss mit beachtet werden, um besser optimieren zu können. Weitere Betrachtungen drehen sich um Redundanz, d. h. dass Ausfälle einzelner Links trotzdem vom Netz verkraftet werden können.

Für Path Diversity kann man ein n_d vorgeben, auf wie viele Pfade ein Fluss mindestens aufgeteilt werden muss, um Lastverteilung zu erzwingen. So kann bspw. erzwungen werden, dass mind. 5 Pfade benutzt werden.

4.4 Lower Bounded Flows

Die bisherigen Probleme waren immer lineare Optimierungsprobleme. Die nächsten Probleme inklusive Lower Bounded Flows beschäftigens sich mit ganzzahligen Problemen.

Es ist praktisch nicht sinnvoll, dass jeder kleine Link mitgenutzt wird, da man so nur die Ausfallwahrscheinlichkeit erhöht, ohne signifikante Durchsatzverbesserungen zu erzielen. Stattdessen möchte man möglicherweise fordern, dass für jeden Demand eine Untergrenze eingehalten wird, wie groß der transportierte Anteil sein muss. Die dafür eingeführten Binärvariablen, ob ein Pfad genutzt wird oder nicht, macht jedoch das Problem ganzzahlig und erfordert Verfahren wie Branch and Bound, um das Problem zu lösen. Das heißt jedoch noch nicht, dass das Problem an sich schwierig ist (sondern vielleicht nur die Lösungsmethode)!

4.5 Limited Demand Split

Betrachte nun den Extremfall von LBF, bei dem eine Single Path Allocation gefordert wird. Dieses Problem ist dann NP-vollständig und entsprechend ist kein Verfahren bekannt, dieses Problem in polynomieller Zeit zu lösen.

Man kann die Constraints jetzt so ansetzen, dass man bspw. eine exakte Anzahl Pfade für jeden Demand gewählt werden. Das schränkt die Optimierung jedoch stark ein. Gleichermaßen können zu entspannte Constraints dafür sorgen, dass bspw. nur kleine Anteile über viele Pfade gehen, während der größte Anteil (praktisch alles) auf einen Pfad geleitet wird. Alternativ kann man aber die Anzahl Pfade, über die der Demand aufgeteilt wird, nach oben begrenzt werden.

4.6 Modular Flow Allocation

Die Linkkapazitäten sind normalerweise in festgelegten Größen verfügbar. Es ist nicht möglich, nur einen Bruchteil einer solchen Kapazität zu allokieren. Bspw. werden nicht 17.3 *Gbps*, sondern 20 *Gbps*, 40 *Gbps*, etc. verschickt (also die Größen, in denen auch Links von Providern angeboten werden).

Für MFA werden die Demands in *demand modules* aufgeteilt, die alle eine Kapazität L_d haben. Die Gesamtkapazität eines Demands ergibt sich dann durch Multiplikation.

4.7 Modular Links

Linkgrößen werden i. d. R. nur in festen Größen angeboten. Entsprechend muss hierfür die Linkdimensionieren angepasst werden, sodass immer mindestens so viele Module ausgewählt werden, dass deren Gesamtkapazität ausreichend für die Linklast ist.

Wenn man jetzt dabei noch die Kosten minimieren möchte, muss das Optimierungsproblem umformuliert werden, sodass y_e nur noch ganzzahlig ist. Entsprechend ist das Problem nur schwer exakt zu lösen. Es bietet sich also hier wieder intuitiv an, Heuristiken zu benutzen. Allerdings kann dies schnell dazu führen, das Optimum weit zu verfehlen. Es stellt sich heraus, dass das Problem NP-vollständig ist.

Das Problem lässt sich zudem noch abwandeln, indem Linkmodule verschiedene Linkkapazitäten können (bspw. in der Praxis bei Glasfasern mit WDM realisierbar). Dies wird im Problem LMMS modelliert.

4.8 Convex Cost and Delay Functions

Konvexe Kostenfunktionen können angenähert werden, indem eine lineare Approximation der konvexen Kostenfunktion als Eingabe für den Optimierer benutzt wird. Als Approximation können bspw. Teilintervalle jeweils durch unterschiedliche lineare Funktionen approximiert werden, wie im Beispiel gezeigt wird. Dies funktioniert natürlich nur in einem gewissen Intervall; außerhalb des Intervalls wächst die Originalfunktion dann deutlich schneller als die lineare Approximation. In diesem Bereich funktioniert die Approximation dann nicht mehr!

In der auf (5) gezeigten Approximation ist der Funktionswert für jedes Teilstück der Approximation dann das richtige Geradenstück, wenn der Wert maximal ist. Somit lässt sich für die Minimierung die passende Approximation auswählen und der Zielfunktionswert passt dann auch. Da der Optimierer immer Eckpunkte des Lösungsraums als optimale Lösungen findet, ist die approximierte Lösung an diesen Punkten dann auch eine optimale Lösung. Dadurch ist jedoch trotzdem nicht bekannt, wie weit der optimale Wert der Approximation vom tatsächlichen Optimum der Originalfunktion entfernt ist. In der Praxis zeigt die feingranulare Wahl von s_k gute Ergebnisse.

Diese Problem lassen sich jetzt auch wieder auf mehrere Links ausweiten.

Manchmal ist auch nur die Steigung interessant. Hierbei kann man sich die Konvexität zunutze machen, da jetzt bekannt ist, dass die Steigungen in aufsteigender Reihenfolge immer mit $<$ geordnet sind (d. h. a_K ist das größte a_k für $k \in \{1, \ldots, K\}$). Es bietet sich also an, jedes z_i größtmöglich zu wählen, da man sonst mit den nachfolgenden z_i in Bereichen größerer Kostenanstiege liegt.

4.9 Concave Link Dimensioning Functions

Konkave Funktionen haben im Gegensatz zu konvexen Funktionen immer weiter sinkenden Anstieg. In unseren Modellen wird zusätzlich noch gefordert, dass der Anstieg positiv bleibt. Solche Kostenfunktionen können bspw. bei Hardwarekosten entstehen (d. h. bessere Hardware hat einen kleineren Aufpreis).

Hier bestehen ähnliche Probleme wie für konvexe Funktionen. Zusätzlich gibt es hier jedoch auch das Problem, dass es viele lokale Minima gibt. Wir können dennoch eine ähnliche Strategie wie für konvexe Funktionen benutzen. Im Gegensatz zu den konvexen Funktionen können die Probleme für konkave Funktionen nicht als reine LOPs gelöst werden, sondern benötigen Ganzzahl-Constraints.

Eben ließ sich der „richtige“ Bereich für die z_k noch recht einfach bestimmen. Das ist hier jedoch nicht möglich. Dafür wird jetzt in den Constraints gefordert, dass u_k immer nur einmal $= 1$ sein darf. Zusätzlich werden noch Hilfsvariablen y_k eingeführt, die aufsummiert das y ergeben und in ihrem Wert durch Δu_k nach oben beschränkt sind.

Das führt dazu, dass genau ein y_k ausgewählt wird, welches dann auch den kleinsten Wert hat. Alle anderen y_k bleiben $= 0$.

4.10 Budget Constraints

Kernidee: mache so viel Durchsatz wie möglich, ohne ein Budget zu sprengen. Es ist dabei insbesondere möglich, mehr als vom Kunden gefordert bereitzustellen.

4.11 Incremental Network Design Problems

Kernidee: Gegeben ein Netz mit bestehenden Kapazitäten, wie kann ich mein Netz so umbauen, dass mit möglichst wenig Aufwand zusätzliche Kapazitäten/Demands bereitgestellt/erfüllt werden können. Die vorhandenen Kapazitäten sollen also weiter benutzt werden, aber neue Kapazitäten sollen hinzukommen.

Dabei zeigt sich, dass die Bestandskapazitäten c_e konstant sind und damit keinen Einfluss auf die Variablenbelegung haben. Diese können also theoretisch für die Berechnung der Zielfunktion ignoriert werden. Dadurch ergibt sich eine Reduktion der Freiheitsgrade und damit auch eine verschlechterung der optimalen Lösung.

4.12 Representing Nodes

Bisher wurden nur Kantenkapazitäten betrachtet. Knoten wurden als kostenlos und fehlerfrei angesehen, was jedoch in der Realität nicht der Fall ist. Knotenkapazitäten und -kosten können repräsentiert werden, indem aus jedem Knoten je ein Knotenpaar mit einer verbindenden Kante erzeugt wird. Alle eingehenden Kanten gehen auf dem hinteren Knoten ein, alle ausgehenden Kanten gehen vom ausgehenden Knoten aus. Dann können Kosten des Knotens einfach als Kantenkosten der verbindenden Kante zwischen den beiden Knoten modelliert werden.

Für ungerichtete Graphen hingegen werden künstlich Eigenkanten eingeführt, die vom Knoten zu sich selbst gehen und auf der die Kosten des Knotens modelliert werden. Diese künstliche Kante lässt sich dann wieder in Pfade und somit auch das Optimierungsproblem einfügen.

5 Network Resilience

5.1 Depth First Search

- Präorder: -1 ist default für alle Knoten, zeigt an, dass der Knoten noch unmarkiert ist. Der Wert zählt, der wievielte Knoten das ist, zu dem hingegangen worden ist, aka. wann wird welcher Knoten betreten.
- Postorder: Wird nach der Rekursion gesetzt. Der zählt, der wievielte Knoten das ist, zu dem zurückgekehrt worden ist, aka. wann wird welcher Knoten verlassen.

Postorder wird hier nicht weiter benötigt, sondern nur Präorder.

Später bei der Klassifikation werden teilweise wieder alle Kanten betrachtet! Eine Baumkante gehört zum Tiefensuchbaum. Eine Rückwärtskante geht von einem Knoten zu einem anderen Knoten im Baum, der im Tiefensuchbaum näher an der Wurzel und im gleichen Zweig ist. Alles andere sind Kreuzkanten. Letztere können in einem Tiefensuchbaum nicht auftreten.

5.2 Verifying Descendant Relationships via Preordering

Damit lässt sich prüfen, ob wir noch Netz über einen Großvaterknoten bekommen können, wenn ein Vaterknoten stirbt. Somit lässt sich also ermitteln, welche Knoten Artikulationsknoten sind.

Artikulationsknoten sind genau die v, die keine Blätter sind, und für irgendeinen Unterbaum von v kein Kindknoten von v eine Rückwärtskante zu einem echten Vorfahren von v hat. Es gibt dann also Kindknoten von v, deren Rückwärtskanten höchstens zu v, jedoch nicht weiter Richtung Wurzel gehen.

5.3 Low Value of Nodes

$low(v)$ lässt sich durch den Präorder-Wert von v sowie den *low*-Werten der *direkten* Kindknoten sowie einiger ihrer Präorder-Werte zu berechnen.

Der Wert gibt den Präorder-Wert des kleinsten Knotens an, zu dem man von dem Knoten hin über Rückwärtskanten zurückgehen kann. *low* lässt sich während der Durchführung der Tiefensuche rekursiv berechnen.

Ein Knoten v (der nicht die Wurzel ist) ist genau dann Artikulationsknoten, wenn für irgendein Kind der *low*-Wert größer oder gleich $v.pre$ ist. Sonst würde es einen Pfad von einem Kind zu einem Knoten oberhalb von v geben. Wurzelknoten sind genau dann Artikulationsknoten, wenn sie mehr als einen Kindknoten haben, da genau dann der Weg

über die Wurzel der einzige Weg von einem direkten Kind zu einem anderen ist (sonst hätte die DFS das eine direkte Kind als direktes Kind des anderen direkten Kindes gewählt – gleiche Argumentation wie für Kreuzkanten). Auch der Test, ob ein Knoten Artikulationsknoten ist, lässt sich in die Tiefensuche (direkt nach der Berechnung von *low*) einbauen.

Stichwortverzeichnis

capacity, 5, 6
cost, 7, 8

demand, 6
    directed, 6
    undirected, 5
dual, 13
duality, 12

flow, 5, 6
flow allocation vector, 7
flow vector, 7

Link
    undirected, 5
link, 6
    directed, 6
link-path incidence relation, 7
load, 7

multi-commodity flow problem, 5

node, 6

objective, 5

path, 6
    n-hop, 6
    candidate, 6
primal, 13

simplex tableau, 10