Algorithmic Aspects of Communication Networks

Chapter 4 Network Design Problems

Recall our Simple Design Problem (1)

Simple Design Problem
indices

$d = 1, 2, ..., D$	demands
$p = 1, 2, ..., P_d$	candidate paths for flows realizing demand d
$e = 1, 2, ..., E$	links

constants

δ_{edp}	$= 1$, if link e belongs to path p realizing demand d; 0, otherwise
h_d	volume of demand d
ξ_e	unit(marginal) cost of link e

(we will not repeat the meaning of indices every time)

Recall our Simple Design Problem (2)

Simple Design Problem (continued)
variables

x_{dp}	flow allocated to path p of demand d (continuous non-negative)
y_e	capacity of link e (continuous non-negative)

objective

$$\text{minimize } F = \sum_e \xi_e y_e \qquad \text{(bandwidth cost)}$$

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D \qquad \text{(demand constraints)}$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \le y_e, \qquad e = 1, 2, ..., E \qquad \text{(capacity constraints).}$$

Recall our Simple Design Problem (3)

- Solutions to our optimization problem will have y_e equal to the load on the links, as otherwise we could have lower costs
- We can eliminate the variables y_e for the link load by substituting them in the cost function with the load on the links

$$\mathbf{F} = \sum_e \xi_e \sum_d \sum_p \delta_{edp} x_{dp} = \sum_d \sum_p x_{dp} \sum_e \xi_e \delta_{edp} = \sum_d \sum_p \zeta_{dp} x_{dp}$$

with $\zeta_{dp} = \sum_e \xi_e \delta_{edp}$ denoting the cost of path p for demand d

- This leads us to the following formulation:

SDP-Decoupled Link-Path Formulation (SDP/DLPF)
variables x_{dp} flow variable allocated to path p of demand d
objective minimize $F = \sum_d \sum_p \zeta_{dp} x_{dp}$
constraints $\sum_p x_{dp} = h_d, \quad d = 1, 2, ..., D$

(note that this problem formulation has fewer variables)

Recall our Simple Design Problem (4)

- Actually, the formulation on the last slide represents a set of decoupled optimization problems, that is to minimize the cost of the paths for each of the demands:
  - This is due to the linear structure of the objective function
  - An optimal solution to these problems will allocate all demand on the shortest path
  - If there are multiple equally shortest paths for one demand, then this demand can be arbitrarily split among these paths
- In order to generate solutions to the above problem, we need to specify candidate paths, e.g. using an algorithm for computing the k shortest (simple) paths

Capacitated Problems (1)

- Recall our simple capacitated flow allocation problem

Pure Allocation Problem (PAP)

constants

δ_{edp}	$= 1$ if link e belongs to path p realizing demand d; 0, otherwise
h_d	volume of demand d
c_e	capacity of link e

variables

x_{dp}	flow allocated to path p of demand d (continuous non-negative)

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \leq c_e, \qquad e = 1, 2, ..., E.$$

Capacitated Problems (2)

- There is no objective function to optimize, so in this basic form we are just looking for any solution that satisfies the constraints
- As it may be, that there is no solution, we consider the problem:

PAP – Modified Link Path Formulation

variables

x_{dp}	flow allocated to path p of demand d (continuous non-negative)
z	auxiliary continous variable (of unrestricted sign)

objective

minimize z

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \leq z + c_e, \qquad e = 1, 2, ..., E$$

Capacitated Problems (3)

- For this problem, there is always a solution and if $z^* \leq 0$, then x^*_{dp} represent a solution to the original PAP
- The fact that PAP is a linear programming problem has an interesting implication:

If PAP is feasible, then a solution **x** with at most D + E non-zero flows exists

- By adding non-negative slack variables s_e, we obtain the following linear programming problem in standard form:

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} + s_e = c_e, \qquad e = 1, 2, ..., E$$

- The number of non-zero variables in any basic feasible solution is at most equal to the number of equations, implying the above result

Capacitated Problems (4)

- The book [Pioro, p. 114] even states that this implies that if all $s_e > 0$ in the optimal solution (all links are unsaturated), then exactly D flows suffice (thus a single-path allocation exists)
  - Why does this have to be true?
- In many cases, we do not only want to find a feasible solution, but the best solution according to some objective function:
  - Minimizing z in the above example leads to maximizing the minimum unused capacity (recall that z can be negative, thus the smallest z that allows to satisfy all capacity constraints maximizes the unused capacity over all links)
  - We can also maximize the total unused capacity after flow allocation:

$$\textbf{maximize F} = \sum_e r_e \left(c_e - \sum_d \sum_p \delta_{edp} x_{dp} \right) = \sum_e r_e \left(c_e - \underline{y}_e \right)$$

with r_e denoting the revenue associated with one unit of unused capacity on link e (e.g. 1 for all links if no different treatment is desired)

Capacitated Problems (5)

- We can also consider a variation of a mixed dimensioning/capacitated problem, where we want to dimension link capacities y_e but subject to some given upper bounds c_e

Bounded Link Capacities

objective

$$\text{minimize } F = \sum_e \xi_e y_e$$

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \le y_e, \qquad e = 1, 2, ..., E.$$

$$y_e \le c_e, \qquad e = 1, 2, ..., E.$$

Path Diversity (1)

- Sometimes, we wish to allocate flows in a way that no single path flow x_{dp} carries more than a fraction of the demand (expressed by n_d = the number of different paths among which h_d is to be split)

Path Diversity (PD)

variables

x_{dp} flow allocated to path p of demand d (continuous non-negative)

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \le c_e, \qquad e = 1, 2, ..., E$$

$$x_{dp} \le h_d / n_d, \qquad d = 1, 2, ..., D \quad p = 1, 2, ..., P_d$$

Path Diversity (2)

- If n_d is an integer, it will force the demand d to be split among onto at least n_d different paths
- If we supply candidate paths P_{dp} in a way that all paths belonging to one demand d are link disjoint, we can guarantee that for each link failure a demand d can at most loose $(100/n_d)$% of its volume
- We can also formulate the diversity constraint in a stricter way, so that we can pass arbitrary candidate path lists:

Generalized Diversity (GD)

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \le c_e, \qquad e = 1, 2, ..., E$$

$$\sum_p \delta_{edp} x_{dp} \le h_d / n_d, \qquad d = 1, 2, ..., D \quad e = 1, 2, ..., E$$

Path Diversity (3)

- The modified constraint ensures that no link e of a path P_{dp} caries more than (100 / n_d)% of demand d
  - In the book [Pioro, p. 117] it is stated:

    $$\delta_{edp} x_{dp} \leq h_d / n_d, \quad d = 1, 2, ..., D \quad p = 1, 2, ..., P_d \quad e = 1, 2, ..., E$$

  - Does this suffice?
  - As multiple candidate paths for one demand may use one specific link, we need to sum over all candidate paths:

    $$\sum_p \delta_{edp} x_{dp} \leq h_d / n_d, \quad d = 1, 2, ..., D \quad e = 1, 2, ..., E$$

- One drawback of these formulations is that we obtain a lot of constraints
- The shortest path allocation rule used to obtain solutions to the original dimensioning problem can be used in a modified version:
  - First look up the shortest path for a demand and allocate h_d / n_d to it
  - Then allocate the next fraction h_d / n_d to the next shortest path and so on

Lower Bounded Flows (1)

- Sometimes, we want to restrict the flow over a path from below in order to avoid having a flow to be partitioned among too many paths
  - This is somehow the opposite goal to path diversity when we wanted to restrict flows to a certain maximum value per utilized path
  - In order to do so, we need to model that once a path flow is non-zero, it needs to carry at a lower bounded amount of flow
- For this we need to introduce to our formulation:
  - Constants b_d modeling for each demand d the respective lower bound
  - Binary variables u_{dp} modeling the characteristic “once a path flow is non-zero” (that is they are 1 if the flow is non-zero and 0 otherwise)

Lower Bounded Flows (2)

Lower Bounded Flows (LBF)

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \leq c_e, \qquad e = 1, 2, ..., E$$

$$x_{dp} \leq h_d u_{dp} \qquad d = 1, 2, ..., D \;\; p = 1, 2, ..., P_d$$

$$b_d u_{dp} \leq x_{dp} \qquad d = 1, 2, ..., D \;\; p = 1, 2, ..., P_d$$

- The two last constraints enforce that:
  - $x_{dp} > 0$ if, and only if, $u_{dp} = 1$ and
  - the non-zero flows are bounded from below by constant b_d
- However, the binary variables make the problem “difficult” to solve (no methods significantly different from trying out all combinations)

Limited Demand Split (1)

- Sometimes, we want to directly give a limit k_d for the maximum number of non-zero path flows x_{dp} among which a demand d is split
  - One example for this is the single-path allocation (non-bifurcated flows) with all $k_d = 1$
  - This calls for the introduction of binary variables u_{dp} that enforce all but one path flow for each demand to be 0

Single-Path Allocation (SPA)

constraints

$$x_{dp} = h_d u_{dp}, \qquad d = 1, 2, ..., D \;\; p = 1, 2, ..., P_d$$

$$\sum_p u_{dp} = 1, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \leq c_e \qquad e = 1, 2, ..., E$$

Limited Demand Split (2)

- Remarks:
  - The second constraint enforces that for each demand exactly one binary variable u_{dp} is 1 and all others are 0
  - The first constraint enforces that all flows x_{dp} for which u_{dp} is set to 0 are also 0, and that the flow x_{dp} for which u_{dp} is set to 1 satisfies the whole demand h_d
- Difficulty of the Problem:
  - Unfortunately, the problem SPA is known to be NP-complete
- Proof idea:
  - In order to prove this, we will first consider a more basic problem which is known under the name D2CIF (decision of the 2-commodity integer flow problem) and show its NP-completeness
  - We will then show that the integral flow version of the pure allocation problem (PAP) is NP-complete in the special case of unit demands (all $h_d = 1$)
  - This directly implies the NP-completeness of SPA

Recall the CNF-SAT Problem from Complexity Theory

- A boolean formula f consisting of n variables $\{x_1, \ldots, x_n\}$ is in **conjunctive normal form** if it is the conjunction (logical and) of clauses, where each clause C_j is a disjunction (logical or) of variables x_i either in simple (x_i) or negated form ($\overline{x_i}$)
  - Example: $f(x_1, x_2, x_3) = (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2)$
- The problem CNF-SAT (conjunctive normal form – satisfiability) is the problem to decide, if for a given function f there exists a consistent assignment of the values **true** and **false** to the variable x_i so that the value of f under this assignment becomes true
  - Consistent assignment means that all instances of a given variable x_i in the formula must be assigned the same value true or false
- The problem CNF-SAT has been shown to be in NP
- S.A. Cook showed in 1971 that every problem in NP reduces to CNF-SAT (thus CNF-SAT is NP-complete)

(For proofs of these results please see lectures on Complexity Theory)

The D2CIF Problem (1)

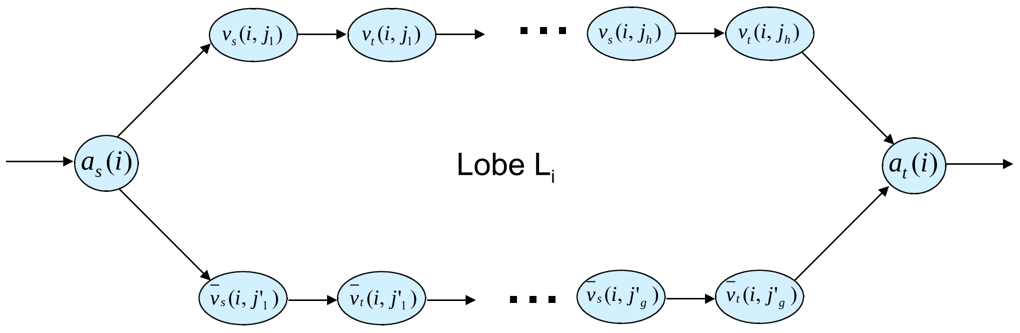
- The D2CIF problem is defined as follows:
  - Given a graph $G(V, E)$ with capacities $c(e) \in \{0, 1\}$ for all edges $e \in E$,
  - Four special nodes: sources S_1, S_2, and targets T_1, T_2
  - Two requirements (demands) R_1, R_2
  - The D2CIF is the decision problem asking whether two-commodity flows, f_1 from S_1 to T_1 and f_2 from S_2 to T_2 exist, such that the total flow $F_1 \geq R_1$ and the total flow $F_2 \geq R_2$ and all x_{1p} and x_{2p} are integers
- Theorem: CNF-SAT reduces to D2CIF
- Proof:
  - We have to show that every CNF-SAT problem can be reduced in polynomial time to a D2CIF problem
  - Consider a CNF expression with n variables $\{x_1, \ldots, x_n\}$ and thus 2n literals $\{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$, which are incorporated in m clauses $C_1, \ldots, C_m$
  - Recall that CNF-SAT for this expression is true, if and only if, there exists a consistent assignment of values true/false to all x_i and respective false/true values to all $\overline{x}_i$ such that there is at least one literal assigned true in each of the m clauses

The D2CIF Problem (2)

- Proof (continued):
  - We will now construct a D2CIF problem that exactly mimics these two requirements:
    - If there is a flow f_1 from S_1 to T_1 with total flow $h_1 = 1$ then the assignment is consistent
    - If there is a flow f_2 from S_2 to T_2 with total flow $h_2 = m$ then all m clauses have at least one literal that is true
    - We will create the flow network in a way, that whenever the flow f_1 runs through a node, the flow f_2 can not run through the same node by splitting the nodes into two nodes v_s and v_t which are connected via a (bottleneck) link of capacity 1
    - Thus, if a solution for the D2CIF is found, the flow f_1 runs through those nodes representing the negated value of the variables x_i
    - A solution for CNF-SAT can then be easily obtained by setting all variables to the negated values of the representing nodes along the path of flow f_1 (e.g. if f_1 flows through $v_s(i, j)$ then set x_i = false, if f_1 flows through $\overline{v}_s(i, j)$ then set x_i = true)

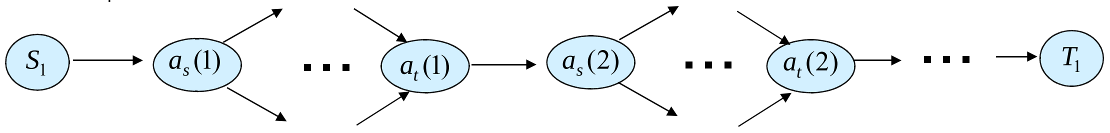
The D2CIF Problem (3)

- For every variable x_i we create a so-called lobe L_i that consists of a start node $a_s(i)$ which models a choice of true/false for the variable x_i and that is connected to a series of:
  - Node tuples $v_s(i, j) \rightarrow v_t(i, j)$ representing either all "positive" instances of x_i in clauses j, or
  - Node tuples $\overline{v}_s(i, j') \rightarrow \overline{v}_t(i, j')$ representing all "negated" instances of x_i in clauses j'
  - The last nodes $v_t(i, j_h)$ and $\overline{v}_t(, j'_g)$ are connected to a merging node $a_t(i)$


$v_s(i, j_1)$
$v_t(i, j_1)$
$v_s(i, j_h)$
$v_t(i, j_h)$
$a_s(i)$
Lobe L_i
$a_t(i)$
$\overline{v}_s(i, j'_1)$
$\overline{v}_t(i, j'_1)$
$\overline{v}_s(i, j'_g)$
$\overline{v}_t(i, j'_g)$

The D2CIF Problem (4)

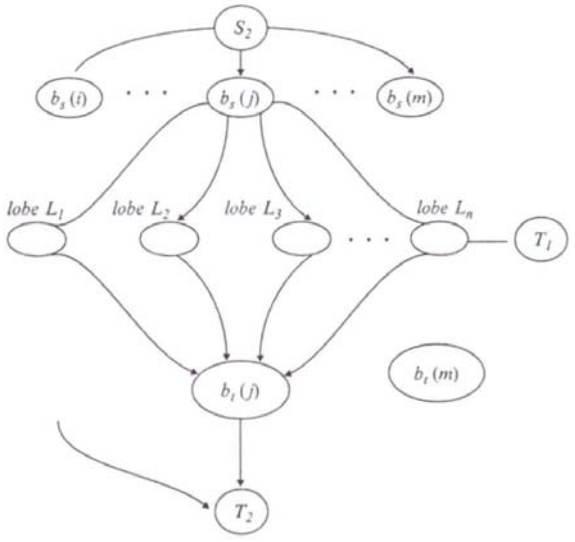
- Now, we need to connect all lobes:
  - We create a start node S_1, connect this to the start node $a_s(1)$ of the first lobe L_1,
  - connect all end nodes $a_t(i)$ of lobe L_i to the start node $a_s(i+1)$ of lobe L_{i+1} (except for the last lobe L_n), and
  - connect the end node $a_t(i_k)$ of the last lobe L_n to a newly created end node T_1



- As all link capacities are set to 1, a flow f_1 from S_1 to T_1 of capacity 1 can take one of 2^n potential paths through the network
- We now have to enforce that such a flow is only possible if variables in all clauses have been set consistently to the respective negated value

The D2CIF Problem (5)

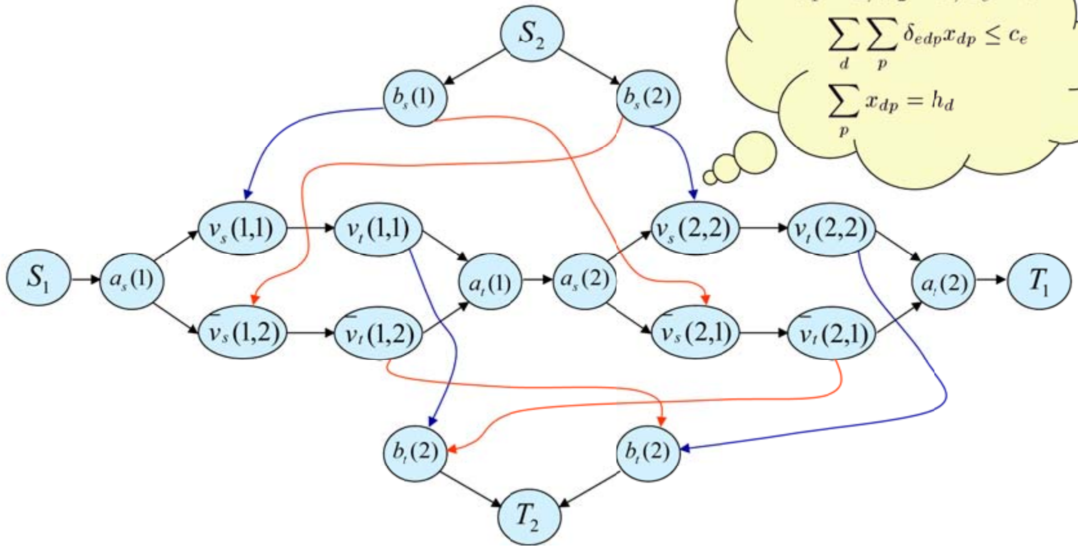
- For this, we create a new start node S_2 and link it to newly created start nodes $b_s(j)$ each representing one clause C_j
- We now link each start node $b_s(j)$ to the appropriate nodes $v_s(i, j)$ or $\overline{v}_s(i , j)$, respectively, depending on if the variable x_i occurs "normally" or negated in clause C_j
- The respective end nodes $v_t(i , j)$ or $\overline{v}_t(i , j)$ are connected to newly created end nodes $b_t(j)$
- All $b_t(j)$ are connected to a newly created end node T_2



(Note that there is only one candidate path from each $b_s(j)$ through every lobe L_i)

The D2CIF Problem (6)

- Example CNF: $f(x_1, x_2) = (x_1 \vee \overline{x}_2) \wedge (\overline{x}_1 \vee x_2)$



$$h_1 = 1,\ h_2 = 2,\ c_e = 1$$
$$\sum_d \sum_p \delta_{edp} x_{dp} \leq c_e$$
$$\sum_p x_{dp} = h_d$$

- Note, that whenever flow f_2 flows through an edge $v_s(i, j) \rightarrow v_t(i, j)$ then flow f_1 can not use that edge anymore as all $c_e = 1$

The D2CIF Problem (6)

- Please note:
  - The demand f_2 can only reach the value m if there is a flow of value 1 between all start nodes $b_s(i)$ and end nodes $b_t(i)$ for i = 1, …, m
  - Usually one lobe L_i has multiple tuples $v_s(i , j) \rightarrow v_t(i , j)$ in the upper part as well as multiple tuples $\overline{v}_s(i , j) \rightarrow \overline{v}_t(i , j)$ in the lower part
  - If a “solution attempt” for demand f_2 tries to make use of an inconsistent assignment of values to a variable x_i in distinct clauses j, then the demand f_1 can not be fulfilled anymore, as this utilizes all “path capacity” through the corresponding lobe L_i
  - In order to “read” the result of the corresponding CNF-SAT after the D2CIF has been solved, just look up the path of demand f_1 through the lobes L_i and assign all values of variables to their negated value for variables x_i
- As such a D2CIF can be constructed in polynomial time for every CNF-SAT, we have shown that CNF-SAT reduces to D2CIF and that D2CIF is thus NP-complete ■
- A corresponding result can be shown for undirected graphs

The Integral Flow Pure Allocation Problem

- The integral flow PAP is defined as follows:

Integral Flow PAP
For a set of demands find an integral solution (all x_{dp} are integers) so that capacity constraints of all edges are not exceeded

- Theorem: In the special case of homogenous unit demands (all h_d = 1), the Integral Flow PAP is NP-complete
- Proof: Modify the construction of the D2CIF proof as follows:
  - Instead of having one source node S_2 for demand f_2 with $h_2 = m$, we introduce m new source nodes that are connected with an edge of capacity 1 to S_2 and that each have a demand of $h_{i1} = h_{i2} = \dots = h_{im} = 1$
  - Thus, if we could solve Integral Flow PAP in polynomial time then we could also solve CNF-SAT in polynomial time ■
- Our result for Integral Flow PAP directly implies that the problem SPA of finding non-bifurcated flows is also NP-complete, as the Integral Flow PAP for homogenous demands is a special case of an SPA

Limited Demand Split (3)

- We can simplify our SPA formulation by eliminating the flow variables x_{dp}:

Single-Path Allocation (SPA)

variables

u_{dp} binary variable $\simeq$ flow allocated to path p of demand d

constraints

$$\sum_p u_{dp} = 1, \qquad d = 1, 2, ..., D$$

$$\sum_d h_d \sum_p \delta_{edp} u_{dp} \leq c_e \qquad e = 1, 2, ..., E$$

- For this, we used the equality $x_{dp} = u_{dp} h_d$ and replaced x_{dp} appropriately in our capacity constraints

Limited Demand Split (3)

- The use of binary variables enables us to formulate also that a demand d has to be equally split among k_d candidate paths

Equal Split Among k Paths

additional constants

k_d predetermined number of paths for demand d

constraints

$$\sum_p u_{dp} = k_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \Big(\sum_p \delta_{edp} u_{dp} \Big) h_d / k_d \leq c_e, \qquad e = 1, 2, ..., E.$$

- To see that the two constraints enforce the equal split of flows
  - Consider additional variables x_{dp} and constraints $x_{dp} = h_d \, u_{dp} / k_d$
  - This would lead to usual constraints $\sum_d \sum_p \delta_{edp} x_{dp} \leq c_e \quad e = 1, 2, ..., E$

Limited Demand Split (4)

Arbitrary Split Among k Paths

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_p u_{dp} = k_d, \qquad d = 1, 2, ..., D$$

$$x_{dp} \leq u_{dp} h_d, \qquad d = 1, 2, ..., D \quad p = 1, 2, ..., P_d$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \leq c_e, \qquad e = 1, 2, ..., E$$

- The second and third constraint jointly enforce that for each demand d non-zero flows can be assigned to at most k_d paths
- As in this problem flows x_{dp} can be of different volumes, we can not eliminate the path flow variables x_{dp} like in the two preceding examples

Modular Flow Allocation (1)

- In transport networks, demand volumes are usually given in terms of modular units, e.g.
  - Plesiochronous Digital Hierarchy (PDH) PCM primary trunks (each offering 32 * 64kbit/s = 2048 kbit/s), or
  - Synchronous Digital Hierarchy (SDH) STM-1 (155 MBit/s), STM-4 (622 Mbit/s), STM-16 (2.4 GBit/s), or STM-64 (9.6 GBit/s) trunks
- In such cases, we model demands d as a number H_d of demand modules each with capacity L_d (thus, $h_d = L_d H_d$)
  - Flows x_{dp} are then forced to be either of capacity L_d or 0 (again using binary variables u_{dp})

Modular Flow Allocation (2)

Modular Flow Allocation (MFA)

additional constants

L_d	demand module for demand d
H_d	volume of demand d expressed as the number of demand modules

constraints

$$x_{dp} = L_d u_{dp}, \qquad d = 1, 2, ..., D \quad p = 1, 2, ..., P_d$$

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \le c_e, \qquad e = 1, 2, ..., E$$

- As $x_{dp} = L_d \cdot u_{dp}$ we can eliminate the variables x_{dp} like we did before and obtain the simpler formulation shown on the next slide

Modular Flow Allocation (3)

Modular Flow Allocation (MFA)

additional constants

L_d	demand module for demand d
H_d	volume of demand d expressed as the number of demand modules

constraints

$$\sum_p u_{dp} = H_d, \qquad d = 1, 2, ..., D$$

$$\sum_d L_d \sum_p \delta_{edp} u_{dp} \le c_e, \qquad e = 1, 2, ..., E$$

- If the demand modules are of same size for all demands, this formulation can be further simplified by replacing c_e with c_e / L_d and eliminating L_d from the second constraint

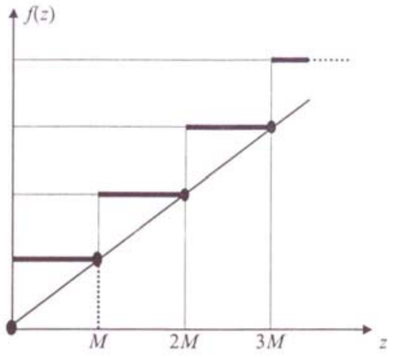
Non-Linear Link Dimensioning, Cost and Delay Functions

- So far, we have mostly assumed that link capacities are equal to the link loads for uncapacitated problems
- Typically, the link cost function is build upon the notion of the link dimensioning function $F_e(\underline{y}_e)$ which determines the relationship between the link load $\underline{y}_e$ and the minimal required link capacity y_e
- Link cost has been computed as capacity times a cost coefficient ξ_e
- Thus, so far we mainly considered linear link cost functions $\xi_e \, \underline{y}_e$
- We now want to extend this by considering also modular links, links with convex cost and links with concave cost

Modular Links (1)

- As we have already seen in our example on modular flow allocation, in practice links are often of modular size
- Let us assume that the size of one link capacity module is M
- The variable y_e denotes the number of link capacity modules
- In this case the link dimensioning function is (note that $\underline{y}_e$, denotes the load on the link e, and is not to be mixed up with the variable y_e):

$$F_e(\underline{y}_e) = k\xi_e \text{ with } (k-1)M \le \underline{y}_e \le kM$$



Modular Links (2)

Modular Links (ML)

additional constants

ξ_e	cost of one capacity module on link e
M	unique size of the link capacity module

objective

$$\text{minimize } \mathbf{F} = \sum_e \xi_e y_e$$

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \le M y_e, \qquad e = 1, 2, ..., E$$

Modular Links (3)

- A heuristic for solving this problem could be assuming a linear approximation of the link dimensioning function $F_e(y_e) = y_e \xi_e$ solving the respective linear programming problem and then to round up the obtained link capacities
- Unfortunately, this may lead to solutions that are far from optimal
- Example:
  - Assume a fully meshed network with V nodes each requesting 1 unit of flow between each pair of nodes D = V · (V - 1) / 2
  - The cost of one module on each link is ξ_e = 1 and the link capacity module M equals to D flow units
  - In this case every optimal solution corresponds to a spanning tree T with y_e = 1 if e $\in$ T and y_e = 0 otherwise
  - The minimal cost of such a solution is V - 1 as every tree has V - 1 links
  - If the above heuristic was applied this would result in a fully meshed network with cost V (V - 1) / 2, thus V / 2 times higher!
  - Thus, it may be cheaper to use longer paths if they have still capacity left

Modular Links (4)

- Difficulty of the problem:
  - The dimensioning problem with modular links ML is NP-complete
- Proof:
  - We will show that the problem can be used to solve the Steiner tree problem (STP) which is known to be NP-complete
  - The STP can be stated as follows: For a Graph G = (V, E), and given a subset V' $\subseteq$ V and link weights ξ_e find a subgraph T $\subseteq$ G so that T contains a path for each pair of nodes in V' and the following cost is minimized:

    $$\xi(T) = \sum_{e \in T} \xi_e$$

  - In other words, STP consists of finding the lightest tree spanning subset V'
  - This solution can (and usually will) contain some but not necessarily all nodes in V \ V' and is thus not equivalent to simply finding the lightest spanning tree in V (which can easily be solved by Prim's or Kruskal's algorithm)
  - To reduce STP to ML, we assign in ML a demand d with volume h_d = 1 to each pair of nodes in V' and set M = V' (V' – 1) / 2

Modular Links (4)

- We can also generalize the ML formulation to cover multiple module sizes M_1, …, M_k where K is the number of module types and variable y_{ek} denotes the number of modules of size M_k installed on link e

Links With Multiple Modular Sizes (LMMS)

additional constants

ξ_{ek}	cost of one capacity module of type k on link e
M_k	size of the link capacity module of type k

objective

$$\text{minimize } \mathbf{F} = \sum_e \sum_k \xi_{ek} y_{ek}$$

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \le \sum_k M_k y_{ek}, \qquad e = 1, 2, ..., E$$

Modular Links (5)

- Note, that modeling K different module sizes increases the number of required variables by a factor of K, as one set of variables is needed for each modular unit type
- Yet another way of introducing modular cost functions with different modules is the incremental characterization:
  - K denotes the number of steps, and
  - The incremental sizes of the link capacity module of type k are modeled with m_1, m_2, …, m_k (if the load on a link passes one of these values, the respective cost function for this link “jumps”)
  - The cost of each incremental module m_k on link e is ξ_{ek}
  - We use binary variables u_{ek} to indicate whether the incremental module of type k is installed on link e (1) or not (0)
- Note that uncapacitated modular design is related to topological design treated earlier, as the first step of the modular capacity function can represent the cost of introducing a new link

Modular Links (6)

Links With Incremental Modules (LIM)

additional constants

ξ_{ek}	cost of one capacity module of type k on link e
m_k	incremental size of the link capacity module of type k

variables

x_{dp}	flow allocated to path p of demand d
u_{ek}	binary variable indicating if module of type k is installed on link e

objective

$$\text{minimize } \mathbf{F} = \sum_e \sum_k \xi_{ek} u_{ek}$$

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \le \sum_k m_k u_{ek}, \qquad e = 1, 2, ..., E$$

$$u_{e1} \ge u_{e2} \ge ... \ge u_{eK}, \qquad e = 1, 2, ..., E$$

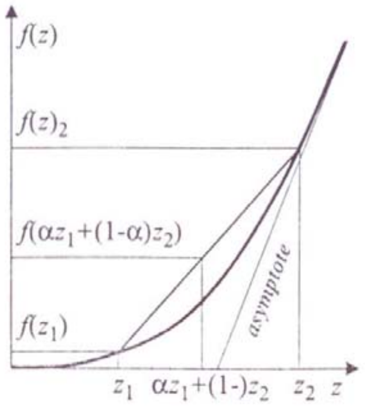
Convex Cost and Delay Functions (1)

- A real-valued function f defined on the interval [0, ∞) is called **convex**, if for any two points $z_1, z_2 \in [0, \infty)$ and any α in [0, 1], we have:

$$\alpha f(z_1) + (1-\alpha) f(z_2) \geq f(\alpha z_1 + (1-\alpha) z_2)$$

- Pictorially, a function is called convex if the function lies below or on the straight line segment connecting two points, for any two points in the interval
- If in the above equation the strong inequality (>) holds for all α in (0, 1) the function is called **strictly convex**
- In networks, convex functions appear to describe delay, e.g. the average delay on a link is a convex function of the link load (recall M/M/1 system):

$$F_e(\underline{y}_e) = \frac{1}{c_e - \underline{y}_e}, \quad 0 \leq \underline{y}_e \leq c_e$$



Convex Cost and Delay Functions (2)

Allocation With Convex Cost Function (CCF)

additional constants

$F_e(\cdot)$ convex cost function of link e

variables

$\underline{y}_e$ load of link e (continous non-negative)

objective

$$\text{minimize } \mathbf{F} = \sum_e F_e(\underline{y}_e)$$

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} = \underline{y}_e, \qquad e = 1, 2, ..., E$$

$$\underline{y}_e \leq c_e, \qquad e = 1, 2, ..., E$$

- Even though the functions $F_e()$ are not constant, we list them under constants to avoid creating yet another category of parameters

Convex Cost and Delay Functions (3)

- If we use the functions $F_e(\underline{y}_e) = \frac{y_e}{c_e - \underline{y}_e}, \quad e = 1, 2, ..., E$
  then the resulting objective is proportional to the average network delay experienced by the packets
  - These cost functions are not meaningful outside the interval $[0, c_e]$
- Convex functions can also be used to convert capacitated flow allocation problems to uncapacitated ones by using **penality functions**
  - We ensure that the penality cost function is convex and incurs a high cost if the link capacity is violated
  - E.g. we define a large link dependent penality coefficient ξ_e and define

$$F_e(\underline{y}_e) = \begin{cases} 0 & \text{if } \underline{y}_e \le c_e \\ \xi_e(\underline{y}_e - c_e)^2 & \text{if } \underline{y}_e > c_e. \end{cases}$$

  - We obtain the uncapacitated problem from CCF by omitting constraints

$$\underline{y}_e \le c_e, \quad e = 1, 2, ..., E$$

Convex Cost and Delay Functions (4)

- Solving convex optimization problems (here: optimizing a convex objective function under linear constraints) requires different techniques than those used for linear optimization problems
- However, if we approximate the convex function with a piecewise linear approximation, we can solve a “corresponding” linear problem:
- Example:

$$f(z) = \begin{cases} 0 & \text{for } 0 \le z \le 1 \\ (z-1)^2 & \text{for } z > 1 \end{cases}$$

- We approximate:

$$g(z) = \begin{cases} 0 & \text{for } 0 \le z \le 1 \\ z - 1 & \text{for } 1 \le z < 2 \\ 3(z-2) + 1 = 3z - 5 & \text{for } 2 \le z < 3 \\ 10(z-3) + 4 = 10z - 26 & \text{for } z \ge 3 \end{cases}$$

Convex Cost and Delay Functions (5)

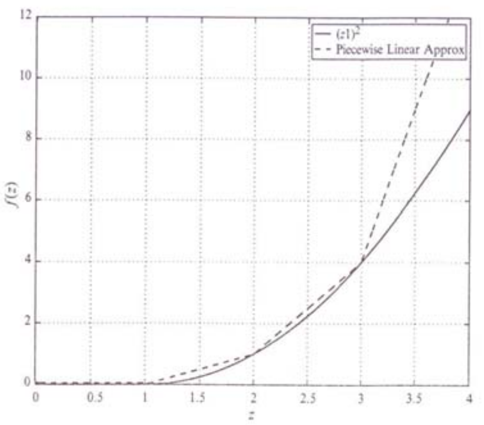
- In the general case, we consider the function defined as:

  $$g(z) = g_k(z) = a_k z + b_k$$

  $$s_{k-1} \leq z < s_k, \quad k = 1, 2, ..., K$$

  with $s_1 = 0$ and $s_K = \infty$
- Suppose, we are given a number $y \geq 0$
- Due to the convexity of g (**!**), the following equation holds:

  $$g(z) = \max_{k=1,2,...,K} \{a_k z + b_k\}$$



- Hence, the optimal solution of the following LP problem will return the correct value for g(y), i.e. r = g(y)

  $$\text{minimize } r$$
  $$\text{subject to } r \geq a_k y + b_k, \quad k = 1, 2, ..., K.$$

Convex Cost and Delay Functions (6)

- As a linear programming problem has its optimum solutions in the edges of the polytope describing the area of valid solutions, the solution computed will be a point y_g^* (= s_i , $0 \leq i \leq K$) where the value of the approximation function is equal to the value of the approximated function: $g(y_g^*) = f(y_g^*)$
- Thus, **an optimal solution** to the approximative problem is **also a valid solution** to the (original) approximated problem
- However, it might be **not an optimal solution** to the original problem
- Approximation error:
  - Let us assume that $y_g^* = s_k$. The maximum approximation error depends on where the original problem has its optimum solution y_f^*
  - It might happen that an approximation with different points s'_i leads to a different optimum solution and there is no simple way (simple enough for this course) to compute the approximation error
- For many practical applications/problems, a fine-grained choice of s_k leads to rather good approximation results

Convex Cost and Delay Functions (7)

Convex Penality Function with Piecewise Linear Approximation (CPF/PLA)

additional indices & constants

$k = 1, ..., K_e$	consecutive pieces of the linear approximation of $F_e(\cdot)$
a_{ek}, b_{ek}	coefficients of the linear pieces of the approximation of $F_e(\cdot)$

variables

r_e continous variable approximating $F_e(\underline{y}_e)$

objective

$$\text{minimize } \mathbf{F} = \sum_e r_e$$

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} = \underline{y}_e, \qquad e = 1, 2, ..., E$$

$$r_e \geq a_{ek}\underline{y}_e + b_{ek}, \qquad e = 1, 2, ..., E \;\; k = 1, 2, ..., K_e$$

Convex Cost and Delay Functions (8)

- Thus, convex mathematical programming problems can be transformed into linear programming (LP) problems
  - If the piecewise linear approximation has the same number K of pieces for every link, the corresponding LP program has E additional variables and E · K additional constraints (compared to the original convex program)
- In many applications, it is not important to know the exact equations of the linear pieces of the approximation
  - Only the slopes a_{ek} and points s_k where they change matter
  - Thus, we obtain a problem like:

$$\begin{array}{ll} \textbf{minimize} & a_1 z_1 + a_2 z_2 + ... + a_K z_K \\ \textbf{subject to} & y = z_1 + z_2 + ... + z_K \\ & 0 \leq z_1 \leq s_1 - s_0 \\ & 0 \leq z_2 \leq s_2 - s_1 \\ & ... \\ & 0 \leq z_K \leq s_K - s_{K-1}. \end{array}$$

Convex Cost and Delay Functions (9)

- Due to convexity we have that $a_1 < a_2 < \cdots < a_K$
  This is why the optimization works.
- If $s_{k-1} \leq y < s_k$ the minimization will set z_1 to s_1, z_2 to $s_2 - s_1$, …, z_{k-1} to $s_{k-1} - s_{k-2}$, and z_k to $y - s_{k-1}$, all remaining z_j will be set to 0
- Thus the problem can be written in a simpler form:

$$\begin{array}{ll} \textbf{minimize} & \sum_k a_k z_k \\ \textbf{subject to} & \sum_k z_k = y \\ & 0 \leq z_k \leq m_k, \quad k = 1, ..., K. \end{array}$$

  with $m_k = s_k - s_{k-1}$ denoting the distance between two consecutive break points of the piecewise linear approximation

Convex Cost and Delay Functions (10)

- We may also face problems with a linear cost function, but convex constraints
- One example is to minimize capacity cost for a fixed routing under the constraint that a given average acceptable delay $\hat{D}$ has to be met
- As the expected delay at a router is a convex function of link utilization, we obtain a convex constraint
- The resulting problem can be solved by various methods:
  - Karush-Kuhn-Tucker conditions (not treated here)
  - Classical Lagrangian multiplier method (not treated here)
  - Piecewise linear approximation as described before

Convex Cost and Delay Functions (11)

Capacity Design With Fixed Routing and Delay Constraint

constants

$\underline{y}_e$	load on link e induced by fixed routing
$\hat{D}$	acceptable delay
H	total traffic volume $H = \sum_d h_d$

variables

y_e	capacity of link e (non-negative continuous)

objective

$$\text{minimize F} = \sum_e \xi_e y_e$$

constraints

$$y_e \geq \underline{y}_e, \quad e = 1, 2, ..., E$$

$$\frac{1}{H} \sum_e \frac{\underline{y}_e}{y_e - \underline{y}_e} \leq \hat{D}$$

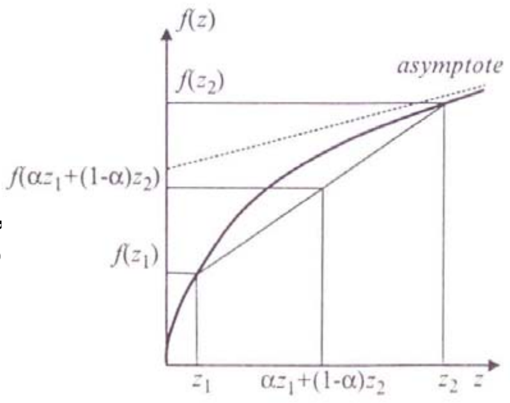
Concave Link Dimensioning Functions (1)

- A real-valued function f defined on the interval [0, ∞) is called **concave**, if for any two points z_1, $z_2 \in [0, \infty)$ and any α in [0, 1], we have:

$$\alpha f(z_1) + (1-\alpha) f(z_2) \leq f(\alpha z_1 + (1-\alpha) z_2)$$

- Pictorially, a function is called concave if the function lies above or on the straight line segment connecting two points, for any two points in the interval
- If in the above equation the strong equality (>) holds for all α in (0, 1) the function is called **strictly concave**
- In networks, concave functions appear to describe link dimensioning functions, as growth in link costs often adheres to the following relation:

$$f(z_1)/z_1 \geq= f(z_2)/z_2 \quad \text{for } z_1 < z_2$$



Concave Link Dimensioning Functions (2)

Concave Dimensioning Functions (CDF)

additional constants

$F_e(\cdot)$ non-decreasing concave dimensioning function of link e

variables

$\underline{y}_e$ load of link e (continous non-negative)

objective

$$\text{minimize } \mathbf{F} = \sum_e \xi_e F_e(\underline{y}_e)$$

constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} = \underline{y}_e, \qquad e = 1, 2, ..., E$$

- In networks concave dimensioning functions arise, for example, when dimensioning telephone networks (computing $\underline{y}_e = F_e(y_e)$ from the inverse of the Erlang Loss Formula)

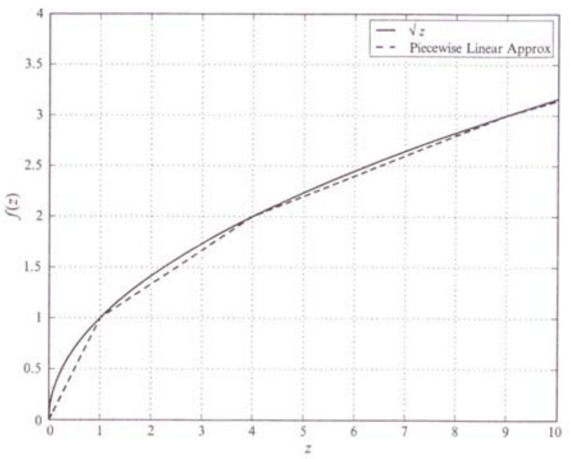
Concave Link Dimensioning Functions (3)

- Because of the property $f(z_1)/z_1 \geq= f(z_2)/z_2$ for $z_1 < z_2$ of the dimensioning function, the optimal solutions to these kinds of problems are non-bifurcated (which is desirable as it enables some heuristics for finding flow allocations; see section 5.6 in the book of Pioro et. al.)
- As these problems require minimizing a concave objective function subject to linear constraints, they in general can have numerous local minima (far away from the global minimum) on the extreme points ("corners") of the feasible region defined by the constraints
- Thus, finding the global minimum can be a very difficult task
- For convex objective functions, we have seen how to transform the problems into linear programming problems by a piecewise linear approximation of the objective function
- Unfortunately, this technique does not work in the same simple way for concave functions, as $g(z) \neq \max_{k=1,2,...,K} \{a_k z + b_k\}$

Concave Link Dimensioning Functions (4)

- However, we can transform the problem into a **mixed-integer programming** approximation
- Consider, for example, the concave square root function and a piecewise linear approximation consisting of four pieces:

$$f(z) = \sqrt{z}, \quad z \geq 0$$



$$g(z) = \begin{cases} z & \text{for } 0 \leq z < 1 \\ (z-1)/3 + 1 = z/3 + 2/3 & \text{for } 1 \leq z < 4 \\ (z-4)/5 + 2 = z/5 + 6/5 & \text{for } 4 \leq z < 9 \\ (z-9)/7 + 3 = z/7 + 12/7 & \text{for } z \geq 9. \end{cases}$$

Concave Link Dimensioning Functions (5)

- In general, let g(z) be defined as:

$$g(z) = g_k(z) = a_k z + b_k \quad s_{k-1} \leq z < s_k, \quad k = 1, 2, ..., K$$

- First, consider the following formulation:

$$\begin{aligned} &\textbf{minimize} && \sum_k u_k(a_k y + b_k) \\ &\textbf{subject to} && \sum_k u_k = 1 \\ & && y \text{ non-negative continuous, } u_k \text{ binary.} \end{aligned}$$

- As the approximation g(z) is concave, this formulation returns the correct value of g(y) for any given y ≥ 0
  - By minimizing the above sum, we force that the right piece of the approximation function is selected for computing g(y)
- However, if we treat y as a variable (see next problem), then the objective contains multiplications of two variables (u_k and y) which is not allowed in mixed integer programming problems

Concave Link Dimensioning Functions (6)

- We will thus introduce some auxiliary variables y_k and additional constraints, and use them to avoid these multiplications

$$\begin{aligned}
&\textbf{minimize} && \sum_k (a_k y_k + b_k u_k)\\
&\textbf{subject to} && \sum_k y_k = y\\
& && y_k \le \triangle u_k, \quad k = 1,2,...,K\\
& && y_k \text{ non-negative continuous, } u_k \text{ binary,}\\
& && \Delta \text{ number larger than any potential value y}
\end{aligned}$$

- The additional constraints force that exactly (and the right) one value y_k will be non-zero and equal to y in the optimal solution
  - Note that for convex functions we minimized over all values of pieces $g_k(z)$ at point y, and thus also the maximum of such values which equals g(y)
  - Here, we minimize over the sum of all $g_k(y)$ and we need to force that all but the right g_k do not contribute to the solution

Concave Link Dimensioning Functions (7)

Concave Dimensioning Functions with Piecewise Mixed-Integer Approximation (CDF/PMIA)

additional indices & constants

$k = 1, ..., K_e$	consecutive pieces of the linear approximation of $f_e(\cdot)$
a_{ek}, b_{ek}	coefficients of the linear pieces of the approximation of $f_e(\cdot)$

variables

y_{ek}, u_{ek}	continous/binary variables for link e

objective

$$\text{minimize } \mathbf{F} = \sum_e \sum_k (a_{ek} y_{ek} + b_{ek} u_{ek})$$

constraints

$$\begin{aligned}
&\sum_p x_{dp} = h_d, && d = 1,2,...,D\\
&\sum_d \sum_p \delta_{edp} x_{dp} = \underline{y}_e, && e = 1,2,...,E\\
&\sum_k y_{ek} = \underline{y}_e, && e = 1,2,...,E\\
&\sum_k u_{ek} = 1, && e = 1,2,...,E\\
&y_{ek} \le \Delta u_{ek}, && e = 1,2,...,E \;\; k = 1,2,...,K_e
\end{aligned}$$

Concave Link Dimensioning Functions (8)

- Assuming that the piecewise linear approximation involves the same number K of pieces for every link, the above problem contains:
  - E × K additional continues variables y_{ek}
  - E × K additional binary variables u_{ek}
  - E × (K + 2) additional constraints
- This already indicates that the problem is difficult to solve
  - In fact, there are no algorithms known that are significantly better than the full search in the space of binary variables
- As with approximation of convex functions, in many cases it is not necessary to consider the exact piecewise approximations but only the slopes a_k are of interest and the points s_k where they change (and the functions are identical for all links)
  - In such cases we can obtain a very similar formulation than the one we have obtained before (see next slide)

Concave Link Dimensioning Functions (9)

$$\begin{array}{ll}
\textbf{minimize} & a_1 z_1 + a_2 z_2 + ... + a_K z_K \\
\textbf{subject to} & y = z_1 + z_2 + ... + z_K \\
& u_1 \geq u_2 \geq \cdots \geq u_K \\
& (s_1 - s_0)u_2 \leq z_1 \leq (s_1 - s_0)u_1 \\
& (s_2 - s_1)u_3 \leq z_2 \leq (s_2 - s_1)u_2 \\
& ... \\
& (s_K - s_{K-1})u_K \leq z_{K-1} \leq (s_K - s_{K-1})u_{K-1} \\
& 0 \leq z_K \leq (s_K - s_{K-1})u_K \\
& z_k \text{ continuous}, \ u_k \text{ binary}
\end{array}$$

- The constraints
  - imply that if $u_k = 1$ and $u_{k+1} = 0$ then $u_1 = u_2 = … = u_k = 1$, and
  - force $z_j = m_j$ for all $j < k$, $0 \leq z_k \leq m_k$ and $z_j = 0$ for all $j > k$ (where $m_k = s_k – s_{k-1}$)

Concave Link Dimensioning Functions (10)

- This again can also be rewritten in a more compact form:

$$\begin{aligned}
\textbf{minimize} \quad & \sum_k a_k z_k \\
\textbf{subject to} \quad & \sum_k z_k = y \\
& m_k u_{k+1} \le z_k \le m_k u_k, \quad k = 1, ..., K-1 \\
& 0 \le z_K \le m_K u_K
\end{aligned}$$

  - The requirement $u_1 \ge u_2 \ge \cdots \ge u_K$ is redundant and has thus been dropped from the problem formulation

Budget Constraints (1)

- So far, we have often minimized cost when choosing from a set of feasible solutions
- In many cases, however, it is sufficient or even better just to stay within a given budget constraint and chose a different optimization goal
- Example: throughput optimization
  - We maximize the proportion r, in how far all demands can be satisfied

    $\sum_p x_{dp} \ge r h_d, \quad d = 1, 2, ..., D$
  - The link capacities y_e are variables (capacities should not be exceeded)

    $\sum_d \sum_p \delta_{edp} x_{dp} \le y_e, \quad e = 1, 2, ..., E$
  - Overall cost for all links should stay within a budget B

    $\sum_e \xi_e y_e \le B$

Budget Constraints (2)

Budget Constraint (BC)

additional constants

B	given budget
h_d	reference volume of demand d

variables

y_e	capacity of link e
r	proportion of the realized demand volumes (continuous non-negative)

objective

maximize r

constraints

$$\sum_p x_{dp} \geq r h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \leq y_e, \qquad e = 1, 2, ..., E$$

$$\sum_e \xi_e y_e \leq B$$

Incremental Network Design Problems (1)

- Often networks are not designed from scratch, but have to be extended with additional resources
- In such cases, there are already existing link capacities c_e and the task is to add additional capacities y_e to account for an increase in the demand volume
- The network design problems considered so far can be easily extended to cope with such situations:
  - In the capacity constraints we simply have to add the existing capacities c_e which are constants in the formulation of the optimization problem (see next slide)
- Note, that the optimal solution to an incremental network design problem can be (read: usually is) higher than the optimal solution of a pure network design problem
  - The reason for this is, that the existing capacities c_e can not be changed, so there is less freedom for the optimization

Incremental Network Design Problems (2)

Simple Extension Problem (SEP)

additional constants

c_e	existing capacity of link e
ξ_e	unit cost of link e

variables

y_e	extra capacity of link e on top of c_e

objective

$$\text{minimize } \mathbf{F} = \sum_e \xi_e y_e$$

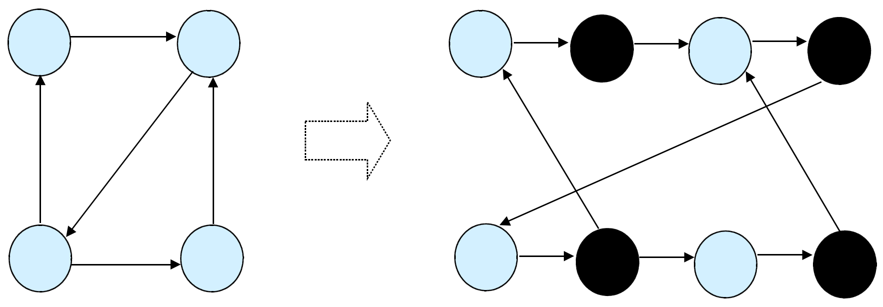
constraints

$$\sum_p x_{dp} = h_d, \qquad d = 1, 2, ..., D$$

$$\sum_d \sum_p \delta_{edp} x_{dp} \leq = y_e + c_e, \qquad e = 1, 2, ..., E$$

Representing Nodes (1)

- In our examples so far, we have only considered link capacities as key network resources and mainly ignored the fact that node capacities can be limited or imply costs as well
- In order to allow for a more realistic modeling, we can help ourselves with the following constructions
- For directed graphs, we split up nodes v into two nodes v' and v'' and introduce a directed internal link (v', v'')
  - Incoming links are connected to v' and outgoing links to v''



Representing Nodes (2)

- For undirected graphs (as in most link-path formulations), we simply introduce additional (articial) links e(v) = (v, v) and define the coincidence coefficients as follows:

$$\delta_{e(v)dp} = \begin{cases} 1 & \text{if node } v \text{ belongs to path } p \text{ for demand } d \\ 0 & \text{otherwise} \end{cases}$$

- The load a node v, defined as $\underline{y}_v = \sum_d \sum_p \delta_{e(v)dp} x_{dp}$

  can be used for various purposes, e.g. to impose an upper bound for the node load:

  $$\sum_d \sum_p \delta_{e(v)dp} x_{dp} \le C_v, \quad v = 1, 2, \ldots, V$$

- The cost of a node v, depending on its load, can appear in the objective function of the design problem

Representing Nodes (3)

- The concept of node-internal links is also useful for modeling node failures: to fail a node, we simply fail its internal link
- If we extend our diversified link-path formulation to encompass node failures as well, then the diversity constraints may get less numerous (but we impose tighter diversity constraints!):

  $$\delta_{e(v)dp} x_{dp} \le h_d / n_d, \quad d = 1, 2, ..., D \quad v = 1, 2, ..., V \quad v \ne s_d, \ v \ne t_d$$

  $$\delta_{edp} x_{dp} \le h_d / n_d, \quad d = 1, 2, ..., D, \quad e = (s_d, t_d) \text{ direct link}$$

  where s_d and t_d denote the source and target nodes of a demand d
- Recall that in the original formulation, we have the following constraints:

  $$\sum_p \delta_{edp} x_{dp} \le h_d / n_d, \quad d = 1, 2, ..., D \quad e = 1, 2, ..., E$$

- Thus, the “node-variant” requires fewer constraints in sparse graphs

Summary

- Formulating network design problems as **linear** or **mixed-integer programming problems** allows for use of the large body of optimization algorithms developed for such problems
- Sometimes even simple looking problems like the **single-path allocation problem** or the **dimensioning problem with modular links** turn out to be **NP-complete**
- Problems with **convex objective functions** or **convex constraints** can be transformed into linear programming problems by **piecewise linear approximation** of the convex functions involved
- Piecewise linear approximation can also by applied for problems with **concave objective functions** or **concave constraints**, however, this leads to **mixed-integer programming problems**
- **Resource restrictions of nodes** can be easily modeled by introducing additional artificial links in case of undirected graphs, and by splitting nodes and introducing additional internal links in directed graphs