Deterministic Discrete Modeling

2b) Deterministic Modelling beyond Formal Logic Part II

Network as Graphs and Traffic Engineering (Optimization Methods)

Dr. Heiko Niedermayer

Cornelius Diekmann, M.Sc.
Prof. Dr.-Ing. Georg Carle

Lehrstuhl für Netzarchitekturen und Netzdienste
Institut für Informatik
Technische Universität München

Version: May 26, 2014

Agenda

1 Networks as Graphs

- Networks and Graphs
- Network Topologies and Topology Generation
- Shortest Paths and Graph Algorithms

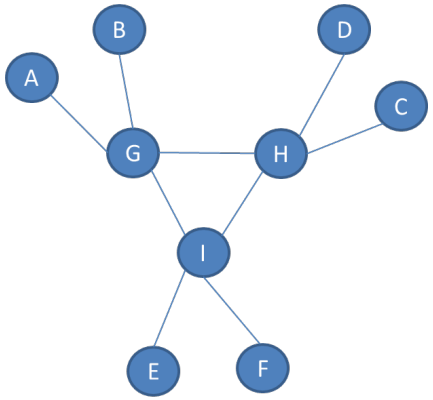
2 Traffic Engineering and Optimization Methods

- Optimization and Convex Optimization
- Shortest Path vs Optimization
- Meta-heuristics / Gradient Descent
- Linear Programming and Dual Problems
- Traffic Engineering: Solving Path-Flow-Problems with LP
- Traffic Engineering: Solving Link-Flow-Problems with LP
- Integer Linear Programming

Slides: 35

Networks as Graphs

Introduction to Graphs



- As you well know a graph `G` consists of `nodes` (●) and `edges` (*l*)
- `G=(V,E)` with `V` the set of nodes (vertices) and `E` the set of edges.
- Edges (A, B) connect two nodes A and B. $A = B \Rightarrow$ self-edge.
- `Undirected Graph:` Edges have no direction.
- `Directed Graph:` Edges (arcs, arrows) have a direction. Edge `(A,B)` from node A to node B can only to be traversed from A to B and not in opposite direction.
- `MultiGraph:` Multiple edges per direction possible.

Some Terminology

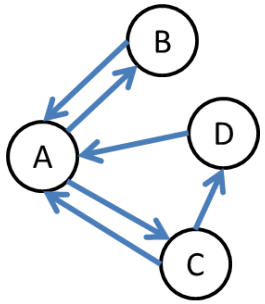
- `Distance:` Length of shortest path
- `Edge Weight:` Distance of an edge (all weights 1 = distance is hop count)
- `Diameter:` Longest distance in graph
- `Strongly-Connected Component:` A subset of nodes and edges that can mutually reach each other.
- `Connected Component:` A subset of nodes and edges that can mutually reach each other if graph were undirected.
- In communication networks, we often implicitly assume a connected graph. This is challenged by NAT and firewalls though.
- `n = Number of nodes`
- `m = Number of edges`

Describing Graphs

- Adjacency List: List of all n nodes and in each line all edges from the node to other nodes are listed.

Example:

```
A B C
B A
C A D
D A
```



- Adjacency Matrix: n-x-n matrix with weight from all nodes to all nodes. 0 means no edge.

```
0 1 1 0
1 0 0 0
1 0 0 1
1 0 0 0
```

- For export and import there are graph formats like GEXF, GML, GraphML, . . . that multiple tools understand.

Networks vs Graphs

- Graphs abstract from the actual network.
- It preserves the virtual structure of the network (relationships between nodes) and algorithms can operate on it.
- Node properties and link properties can be added to graphs as node or edge properties.
- Graphs do not cover aspects of networks where networks use edges depending on policies (e.g. due to peering agreements and implication of link costs), e.g. path via C to A only for own nodes, most likely worse path via D to A for other nodes.

Describing Networks

- The data model of a network contains more information than a graph.
  - Device information
  - Physical and virtual locations of device
  - State information
  - ...
- Management Information Bases (MIBs) in network management are tree-like data structures that can be queried and walked through from the network, e.g. via SNMP.
- Description languages like Network Description Language (NDL) or Network Markup Language (NML) provide XML data formats for network data models.

Python Networkx Basics

Networkx is a graph library for Python, for documentation:
`http://networkx.github.io/documentation/latest/`

To generate a graph:

```
import networkx as nx
G = nx.Graph() # or nx.DiGraph() or nx.MultiGraph()
```

To add nodes:

```
G = G.add_node(name)
G = G.add_node(name, cost=0.3, foo=bar)
```

To add edges:

```
G = G.add_node(sourcenode, destnode)
G = G.add_node(sourcenode, destnode, weight=0.3, foo=bar)
```

Python Networkx Basics

Accessing and Changing properties:

```
G.node[name]['foo'] = bar
G.edge[source][dest]['weight'] = 0.3
```

For more detail, we refer to the library and our example python files. The library will provide functions for many graph properties and algorithms.

Simple Topologies

- Nodes are given. With a `topology`, we mean which edges do exist and what kind of structure does than form.
- Prominent Topologies
  - Full-Mesh: all nodes are mutually interconnected
    - `m = n * (n - 1)`. Half if undirected.
  - Circle: the structure forms a circle or cycle
    - `m = n`
  - Tree: the structure forms a tree.
    - `m = n - 1`

Maps and Topologies

- To model the positions of nodes in the real-world, they could be placed on a map.
- This may allow to estimate the physical distance between nodes and use it in the graph.
- This information may also be used to generate a topology.

Physical Layer / Lower Layer vs Higher Layer Topologies

- On lower layers physical layer proximity is more relevant than on higher layers.
- This is due to the cost of physical (e.g. optical) links that need to be built and maintained.
- The lower layers provide the local connectivity between nodes that then allows the higher layers to cross the Internet.

- Gabriel graphs are one example for a topology that represents more the lower layers like from an Internet Service Provider.
- `http://sndlib.zib.de` provides a large set of topologies and optimization problems from real backbone networks.

Shortest Path Problems

- Shortest Path Problems can be specified in different ways
  - Find shortest path from A to B
  - Find shortest path to all nodes from A (single-source-shortest path)
  - Find shortest path from all nodes to all other nodes (all-pairs shortest path)
- Routing or forwarding in networks does not necessarily want shortest paths
- Finding a cost-optimal spanning tree will not provide shortest paths, but connected network
  - Connectivity vs Shortest Path

Algorithms for Shortest Path Problems

- Dijkstra: greedy algorithm for single-source shortest path, needs non-negative link weights
- Bellman-Ford: all-pairs shortest path algorithms, can handle negative link weights
- Meta Algorithms
  - A*: search strategy, generalization of Dijkstra's algorithm
  - Breadth-First-Search
  - Depth-First-Search

- `nx.shortest_path(G)` will give you a list of shortest paths. Source and destination can be specified.

Traffic Engineering and Optimization Methods

Traffic Engineering

- Traffic Engineering is about optimizing a network for a given traffic demand towards certain goals and constraints.
- This can include
  - Generating the topology
  - Generating the forwarding / routing tables (potentially in different granularity than e.g. IP routing[1], also per flow)
  - ...
  - Question: Is this different from finding the shortest path?

[1] Use optimizer for finding optimal routing, select parameters so that routing protocols (e.g. OSPF) agree under normal circumstances.

Optimization Introduction

- Before answering the question from the previous slide, lets look at optimization.
- In optimization,
  - we have a certain set X of variables x_1, x_2, ... that we can assign to values to.
  - we have a objective function (cost) to determine the cost given certain values for x_1, x_2, ...
  - the goal is to select the values for X so that the objective function is minimized.
  - we tend to have constraints that limit the possible values for X. Depending on the methods, this may be part of the objective function[2]. In others, they have to be specified in addition.
  - the space of all possible values X can take is called search space.

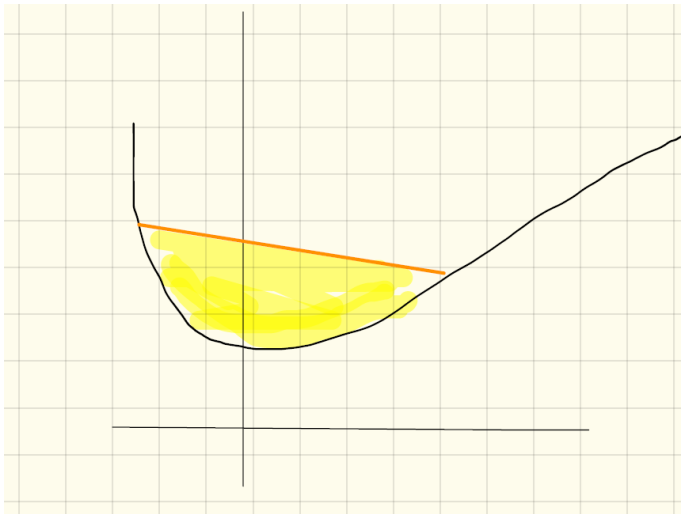
[2] Compare this with the definition of NP as class of problems for which one can verify solutions in P. Here, the objective function verifies if constraints are met and what kind of costs arise in a more or less efficient way. The difficulty is to search for and find the optimal solution.

Objective Function and Convex Optimization

- The objective function (also called fitness, fitness landscape, . . .) can be of any complexity. If it random-looking, it is hard to search for the minimum[3].
- Most search methods (also Genetic / Evolutionary Algorithms) expect that neighboring X generate similar objective function outputs. Due to that, exploring the neighborhood of good values is beneficial.
- Some optimizers expect that the objective function is continuous and differentiable in X.
- The more mathematical properties of an objective function is known, the easier it is to find a global optimum and to proof that this is an optimum.
- If possible, it seems desirable to define optimization problems in way that it is easy to find the optimum.

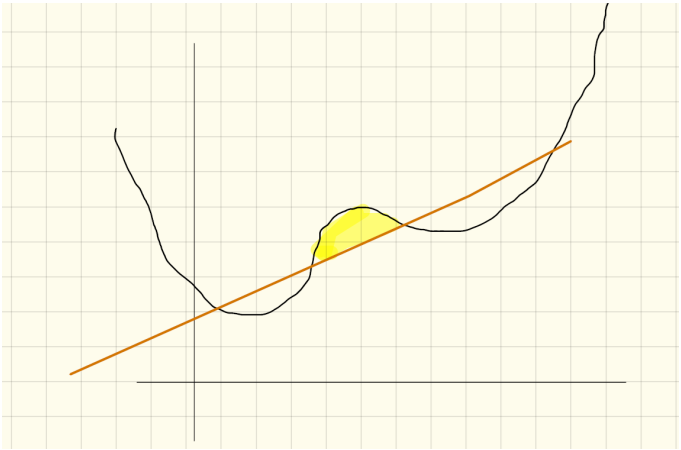
[3] Given the set of all possible functions as objective function, there are so-called No Free Lunch theorems that say that no matter in what order you search the search space, all orderings are the same on average.

Objective Function and Convex Optimization



- Convex Function: a function is convex if any line segment between two points on its graph is equal or above the function.
- Linear functions are convex.
- Quadratic functions like x^2 are convex.
- There are also convex function in multi-dimensional spaces (e.g. $f(x_1, x_2, x_3) = x_1^2 + x_2^2 + x_3^2$).

Objective Function and Convex Optimization



- Assume, we have two minima. The figure above shows that even the smallest peak between these minima will contain points on the graph where the line segment of a line between them will not completely lie over the graph.
- For the traffic engineering problem, we will use the python library cvxopt and the solver glpk in addition, which can be called from cvxopt.

Shortest Path vs Optimization

- Back to the shortest path problem. When is our optimization not the same?

- Shortest Path Problem: Link weights on links, find shortest paths.
- If we add cost to the links and want to minimize cost?
  - Can still be solved by shortest path algorithm, just use cost as link weight.
- If we take the demand into consideration:
  $\sum_{a,b} demand(a,b) * costForPath(a,b)$?
  - Still the same as cost-optimal path is optimized per demand and not inter-demand!

Shortest Path vs Optimization

- When is our optimization not the same?
- Given cost and weight, find shortest paths where cost (different from weight!) is below a threshold.
  - This deviates from a shortest path problem as cost cannot be combined with weight as it is a constraint that we do not optimize but keep below a certain level.
- Given weight, demand, and link capacities, find shortest paths where the demand over the links does not exceed their capcity.
  - This deviates from a shortest path problem as longer paths with more capacity might have to be taken.
- If we sum up different kinds of objectives like link cost per demand over the link and distance for each combination of nodes.
- When we want to ensure certain properties like:
  - k diverse paths per direction
  - avoid certain nodes for certain demands (e.g. Alice and Bob do not want Mallory on their path)

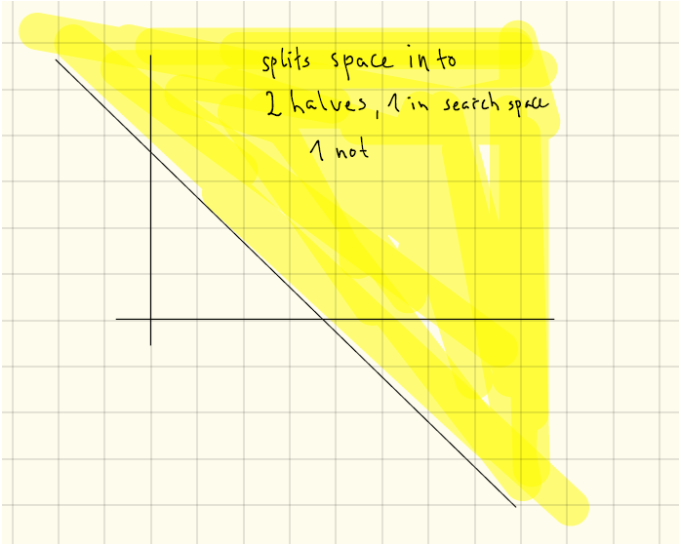
Shortest Path vs Optimization

- Problem: Given cost and weight, find shortest paths where cost (different from weight!) is below a threshold.
- Compute Shortest Paths.
- If cost constraint is met, solution is shortest path.
- If not, loop over until constraints met:
  - Select a set of candidate solutions in the proximity of the current solution.
  - If a better one (with respect to cost, maybe not too much worse than shortest path solution) is found, select it.
  - If cost constraints met, try to optimize distance in similar manner given the cost remains under the threshold.
  - Meta Heuristics:
    - Gradient Descent: Scan the neighborhood of the certain position in the search space, select the one that follows the gradient towards to minimum.
    - In discrete problems, this means scan the neighborhood, go to best solution found there. Then repeat.

Introduction Linear Programming

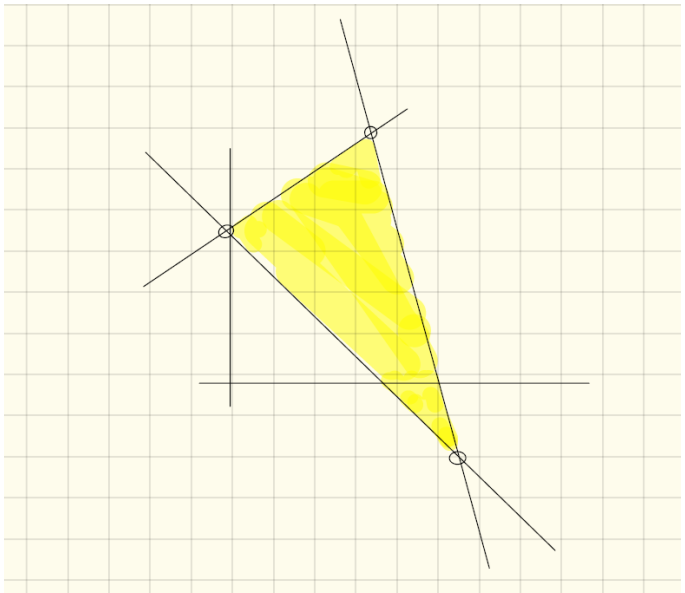
- Linear Programming (Linear Optimization) is a class of optimization problems that are defined solely by linear equations and linear inequalities.
- Objective: Minimize $c^T * x$ where
  - x is a vector of free variables $(x_1, x_2, x_3, \dots)$ to be determined.
  - c is a corresponding cost vector of similar size
  - Objective function: $x_1 * c_1 + x_2 * c_2 + x_3 * c_3 + \dots$
  - Note: Problems to find a maximum can be transformed into problems to find a minimum and vice versa by $-c$.
- Given a linear objective function and no further constraints, the optimimum for an x_i is either $-\infty$ if $c_i > 0$ or ∞ if $c_i < 0$.
- Thus, to have finite optima, there need to be constraints that produce upper or lower bounds for each x_i.

Introduction Linear Programming



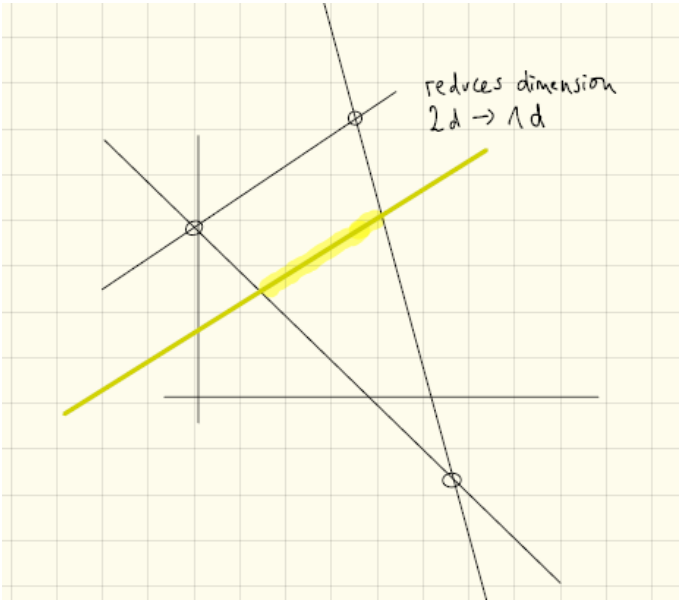
- Linear inequalities ($\leq$ or $\geq$) cut off a part of the search space.
- e.g. $x_1 + 3 * x_2 - 4 * x_3 \leq -3.5$

Introduction Linear Programming



- Given enough inequalities (3 in the figure), the search space can be reduced to a finite area.
- Since x_i's either increase or decrease the cost function, optima have to be on the intersection points on the edge of the search space.

Introduction Linear Programming



- Equalities reduce the search space by 1 dimension. Here from a 2-dimensional area to a line.
- Linear Programs are usually defined by specifying:
  - the cost vector c
  - a matrix G and vector h with $G * x \leq h$ for the inequalities
  - a matrix A and vector b with $A * x = b$ for the equalities

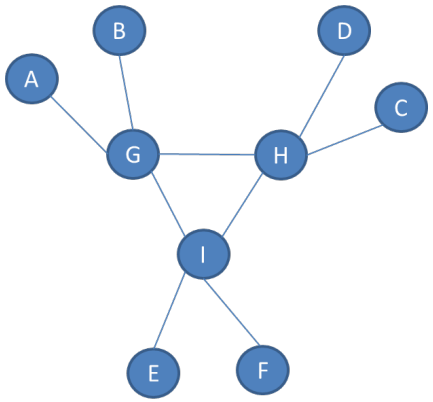
Primal and Dual Problem in Linear Optimization

- The primal problem is the minimization (or maximization) problem given to the linear optimizer.
- The dual problem is then the related maximization problem that helps to compute and proof lower bounds for the minimum. It is not the same problem just written as maximimation problem!

Traffic Engineering

- Network Operators want to dimension that their networks so that the costumer demands are met, the costs are low, the performance good.
- The underlying model is again a graph with routers and other hosts as nodes and the links as edges.
- In addition, some nodes have demands to send certain amounts of traffic to other nodes. This is usually given as traffic matrix.
- Traffic matrix: matrix that contains typical (average, maximum) demands between all nodes.
- The goal is now to find a cost-optimal allocation of the demands over paths in the graph that satisfies all constraints like capacities of links.

Path Flow Problem



- For each demand all (possible, reasonable, desired) paths are given. The problem is to distribute each demand over its possible paths so that the demand is met and costs are optimized.

Path Flow Problem

- Variables x: each x_i stands for one demand and one path for the demand and it says how much of the demand it sent over the corresponding path.
- Cost vector c: $\sum_{demand\ d, path\ p} \sum_{link\ l \in p} x_{indexOf(d,p)}\ cost(l)$
- Inequalities:
  - All traffic over a link l has to be lower than its capacity. $\sum_{x_i\ uses\ l} x_i \leq capacity(l)$
  - No negative traffic amounts. For all x_i: $-x_i \leq 0$
- Equalities:
  - Sum of the traffic sent over the paths for a demand has to match the demand d. $\sum_{x_i\ is\ for\ d} x_i = d$

Link Flow Problem

- In the Path Flow Problem definition we assumed to have a predefined set of paths. We can skip this when we define the x as demand d over a link l.
- Variables x: each x_i stands for one demand and one link and the value is how much of this demand goes over the link.
- Cost vector c: $\sum_{link\ l} \sum_{x_i\ belongs\ to\ l} x_i\ cost(l)$
- Inequalities:
  - All traffic over a link l has to be lower than its capacity. $\sum_{x_i\ uses\ l} x_i \leq capacity(l)$
  - No negative traffic amounts. For all x_i: $-x_i \leq 0$
- Equalities:
  - For each demand and each node, generated / consumed demand by the node has to match outgoing / incoming demand. Routed incoming and outgoing demands have to be equal.

Integer Linear Programming / Mixed Integer Linear Programming

- Integer Linear Programming is a class of linear optimization problems where the variables x are integer numbers instead of real numbers. All other values like cost vector, equlities, and inequalities remain real-valued.
- If some x are integer and some x are real-valued. Then, we speak of a mixed integer linear program.
- In our traffic engineering problems, we need integer linear programs e.g. to disallow the splitting up of demands on parallel paths.

Thanks for your attention!

Questions?