

Algolab BGL – Maximum Flows

(BGL tutorial 2/3)

Alexander Meulemans¹

ETH Zurich

November 2, 2022

¹based on material from Tim Taubner, Stefano Leucci and Daniel Wolleb-Graf

BGL and Maximum Flow

Last BGL tutorial: BGL Intro

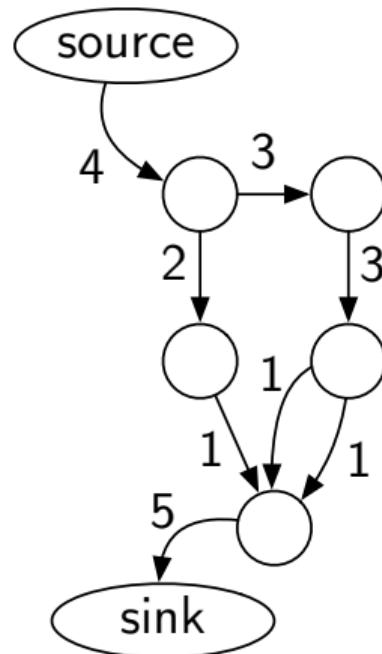
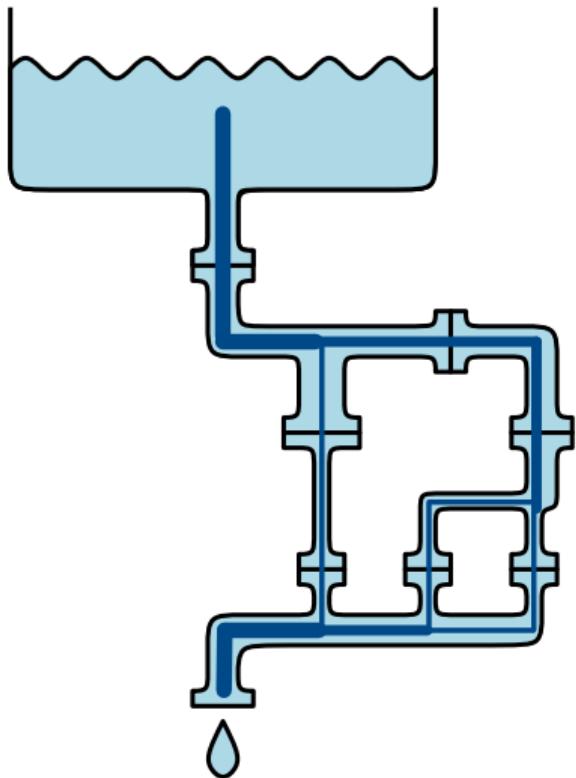
- ▶ Graphs in BGL
- ▶ Standard algorithms: strongly connected components, Dijkstra's shortest paths, Edmond's maximum matching etc.

Today's BGL tutorial: Maximum Flows

- ▶ What is *maximum flow*: motivation, definition and intuition
- ▶ Existing algorithms for maximum flow and how to use them in BGL
- ▶ Common techniques and examples on how to apply maximum flow

Next BGL tutorial: Minimum Cuts, Minimum Cost Maximum Flow, and some more applications

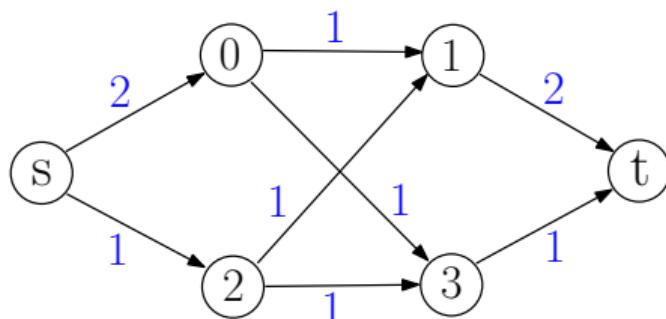
Network Flow: Example



Network Flow: Problem Statement

Input: A flow network consisting of

- ▶ directed graph $G = (V, E)$
- ▶ source $s \in V$ with only out edges
- ▶ sink $t \in V$ with only in edges
- ▶ edge capacity $c : E \rightarrow \mathbb{N}$.



Output: Flow function $f : E \rightarrow \mathbb{R}$ subject to

- ▶ Capacity constraints for every edge e .
 $0 \leq f(e) \leq c(e)$
- ▶ Flow conservation at every vertex u except s and t .

$$\begin{aligned} & \sum_{(v,u) \in E} f((v,u)) \quad \text{in flow} \\ &= \sum_{(u,w) \in E} f((u,w)) \quad \text{out flow} \end{aligned}$$

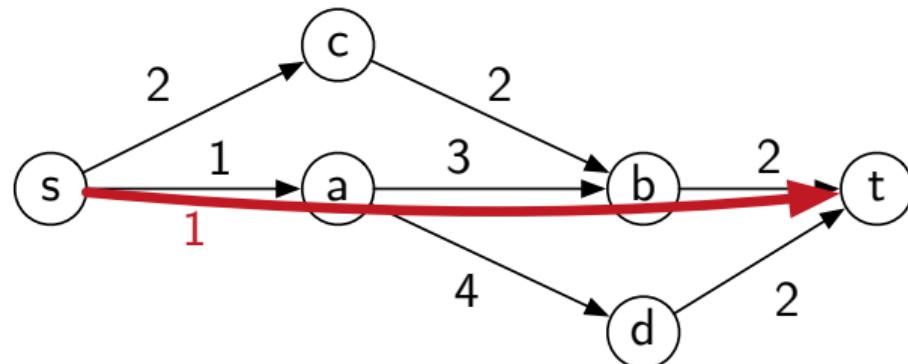
- ▶ Maximization of total flow.
 $|f| := \sum_{(s,v) \in E} f((s,v))$

Theorem

On a graph with integral capacities, there always exists an integral maximum flow.

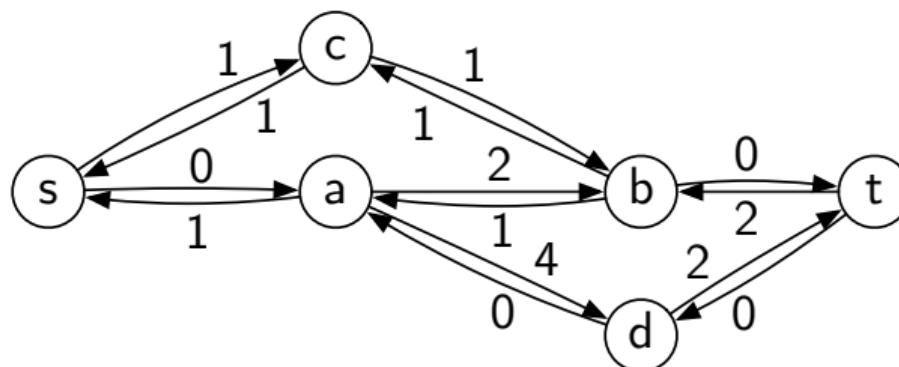
Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- ▶ Take any $s-t$ -path and increase the flow along it.
- ▶ Update capacities and repeat as long as we can.
- ▶ Are we done? Problem: We can get stuck at a local optimum.
Maximal \neq maximum, remember?
"Greedy never works!" (unless you prove it)



Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

- ▶ Solution: Keep track of the flow and allow paths that *reroute* units of flow.
These are called *augmenting paths* in the *residual network*.
- ▶ Ford-Fulkerson (1955): Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- ▶ Edmonds-Karp (1972) / Dinitz (1970): Repeatedly take any *shortest* (as in: fewest edges) augmenting path: running time: best of $\mathcal{O}(m|f|)$ and $\mathcal{O}(m(nm))$ [BGL-Doc].
Simple bound for total flow: $|f| \leq n \max c$



BGL flows: Includes

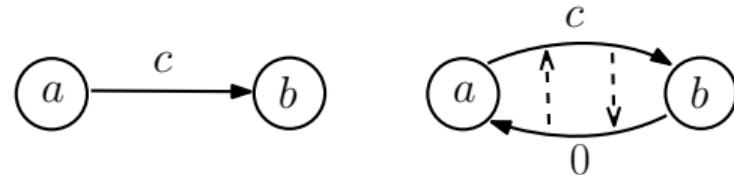
```
#include <iostream>

// BGL include
#include <boost/graph/adjacency_list.hpp>

// BGL flow include *NEW*
#include <boost/graph/push_relabel_max_flow.hpp>
```

Using BGL flows: Typedefs

The typedefs now include residual capacities and reverse edges.



```
// Graph Type with nested interior edge properties for flow algorithms
typedef boost::adjacency_list_traits<boost::vecS, boost::vecS, boost::directedS> traits;
typedef boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, boost::no_property,
    boost::property<boost::edge_capacity_t, long,
    boost::property<boost::edge_residual_capacity_t, long,
    boost::property<boost::edge_reverse_t, traits::edge_descriptor>>> graph;

typedef traits::vertex_descriptor vertex_desc;
typedef traits::edge_descriptor edge_desc;
```

BGL flows: Manually creating an edge

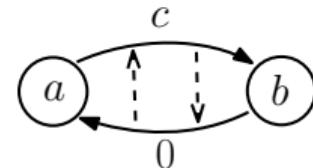
Creating the residual graph: Add a directed edge and its reverse.

```
// Create graph and maps
graph G(4);

auto c_map = boost::get(boost::edge_capacity, G);
auto r_map = boost::get(boost::edge_reverse, G);

vertex_desc a = ...;
vertex_desc b = ...;
long c = ...;

edge_desc e = boost::add_edge(a, b, G).first;
edge_desc rev_e = boost::add_edge(b, a, G).first;
c_map[e] = c;
c_map[rev_e] = 0; // reverse edge has no capacity!
r_map[e] = rev_e;
r_map[rev_e] = e;
```



BGL flows: edge_adder helper class

```
class edge_adder {
    graph &G;

public:
    explicit edge_adder(graph &G) : G(G) {}

    void add_edge(int from, int to, long capacity) {
        auto c_map = boost::get(boost::edge_capacity, G);
        auto r_map = boost::get(boost::edge_reverse, G);
        const auto e = boost::add_edge(from, to, G).first;
        const auto rev_e = boost::add_edge(to, from, G).first;
        c_map[e] = capacity;
        c_map[rev_e] = 0; // reverse edge has no capacity!
        r_map[e] = rev_e;
        r_map[rev_e] = e;
    }
};
```

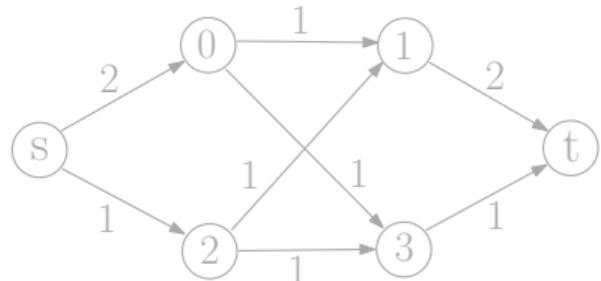
BGL flows: Creating the graph

Graph creation with edge_adder:

```
graph G(4);
edge_adder adder(G);

// Add some edges using our custom edge adder
adder.add_edge(0, 1, 1); // from, to, capacity
adder.add_edge(0, 3, 1);
adder.add_edge(2, 1, 1);
adder.add_edge(2, 3, 1);

// Add special vertices source and sink
const vertex_desc v_source = boost::add_vertex(G);
const vertex_desc v_sink = boost::add_vertex(G);
adder.add_edge(v_source, 0, 2);
adder.add_edge(v_source, 2, 1);
adder.add_edge(1, v_sink, 2);
adder.add_edge(3, v_sink, 1);
```



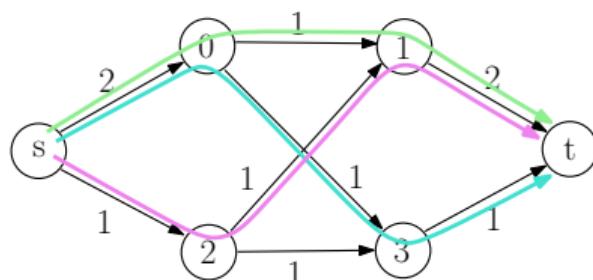
BGL flows: Calling the algorithm

```
// Calculate flow from source to sink
// The flow algorithm uses the interior properties (managed in the edge adder)
// - edge_capacity, edge_reverse (read access),
// - edge_residual_capacity (read and write access).
long flow = boost::push_relabel_max_flow(G, v_source, v_sink);

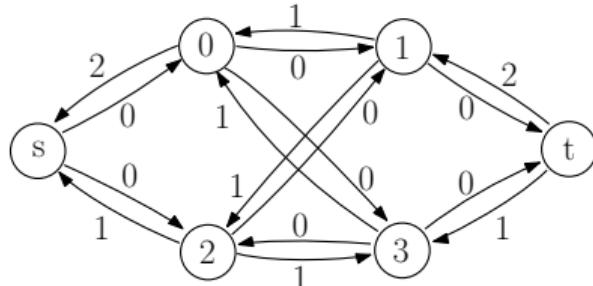
// Retrieve capacity map and reverse capacity map
const auto c_map = boost::get(boost::edge_capacity, G);
const auto rc_map = boost::get(boost::edge_residual_capacity, G);

edge_desc edge = ...;
long flow_through_edge = c_map[edge] - rc_map[edge];
```

Graph with capacities.



Resulting graph with residual capacities.



BGL flows: The different algorithms

Algorithm (BGL-Doc)	by	runtime	use
<code>boost::push_relabel_max_flow</code>	Goldberg, Tarjan (1986)	$\mathcal{O}(n^3)$	almost always
<code>boost::edmonds_karp_max_flow</code>	Edmonds, Karp (1972)	$\mathcal{O}(\min\{m f , nm^2\})$	almost never

Use

```
#include <boost/graph/push_relabel_max_flow.hpp>
...
long flow = boost::push_relabel_max_flow(G, source, target);
```

or

```
#include <boost/graph/edmonds_karp_max_flow.hpp>
...
long flow = boost::edmonds_karp_max_flow(G, source, target);
```

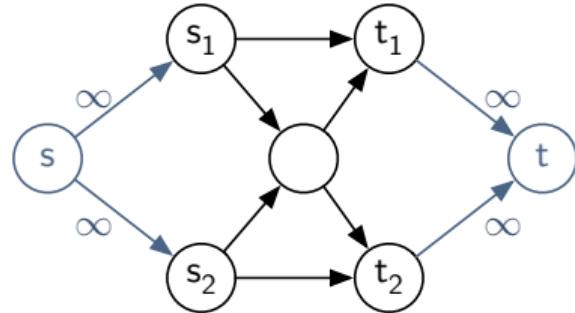
Even for $m, |f| \in \mathcal{O}(n)$, usually push-relabel is at least as fast.

However, it is possible to tailor graphs where Edmonds-Karp beats push-relabel.

Common techniques

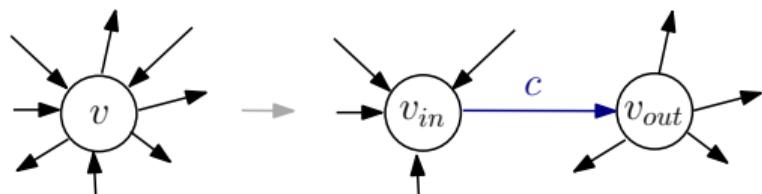
Multiple sources/sinks

with e.g. $\infty \approx \sum_{e \in E} c(e)$



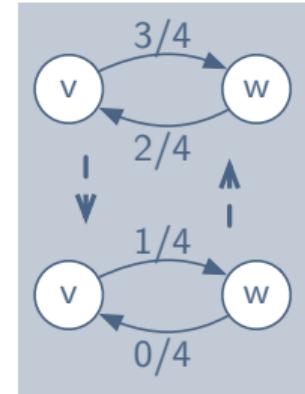
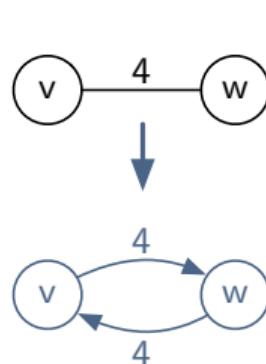
Vertex capacities

split into in-vertex and out-vertex



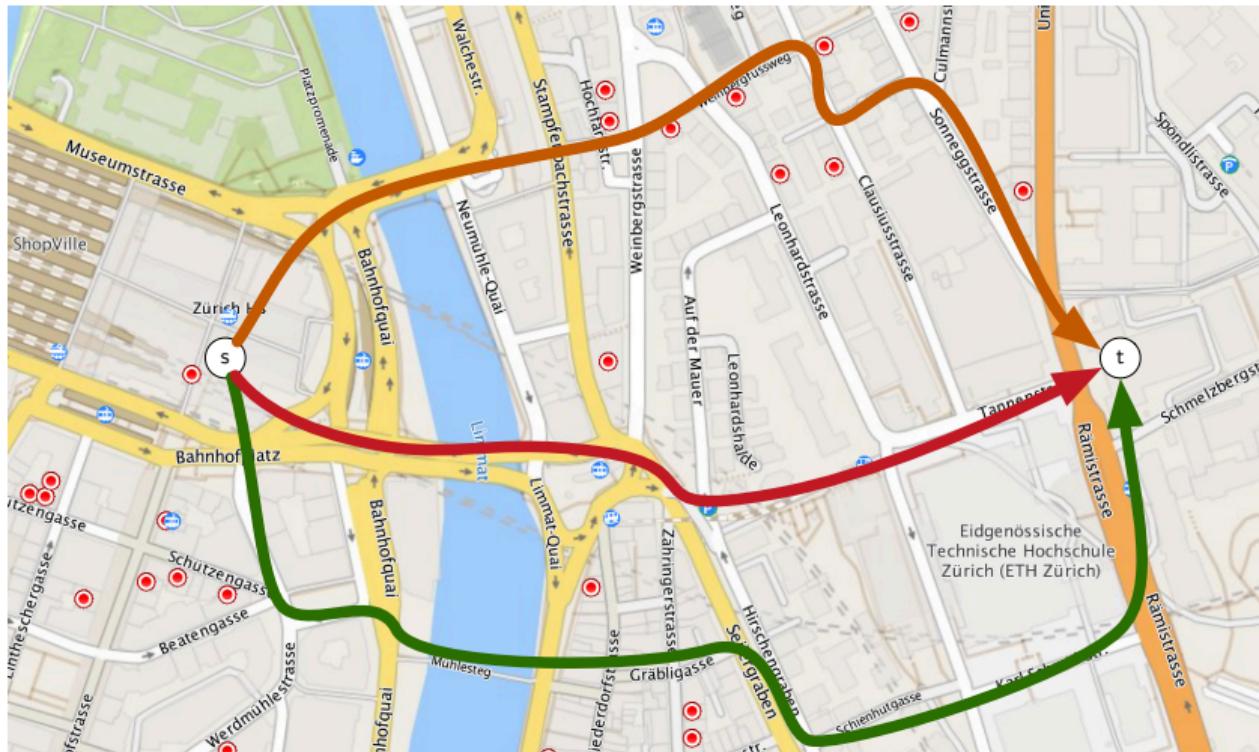
Undirected graphs

antiparallel edges with flow reducable to one direction



Flow Application: Edge Disjoint Paths

How many ways are there to get from HB to CAB without using the same street twice?



Map:
search.ch,
TomTom,
swisstopo,
OSM

Flow Application: Edge Disjoint Paths

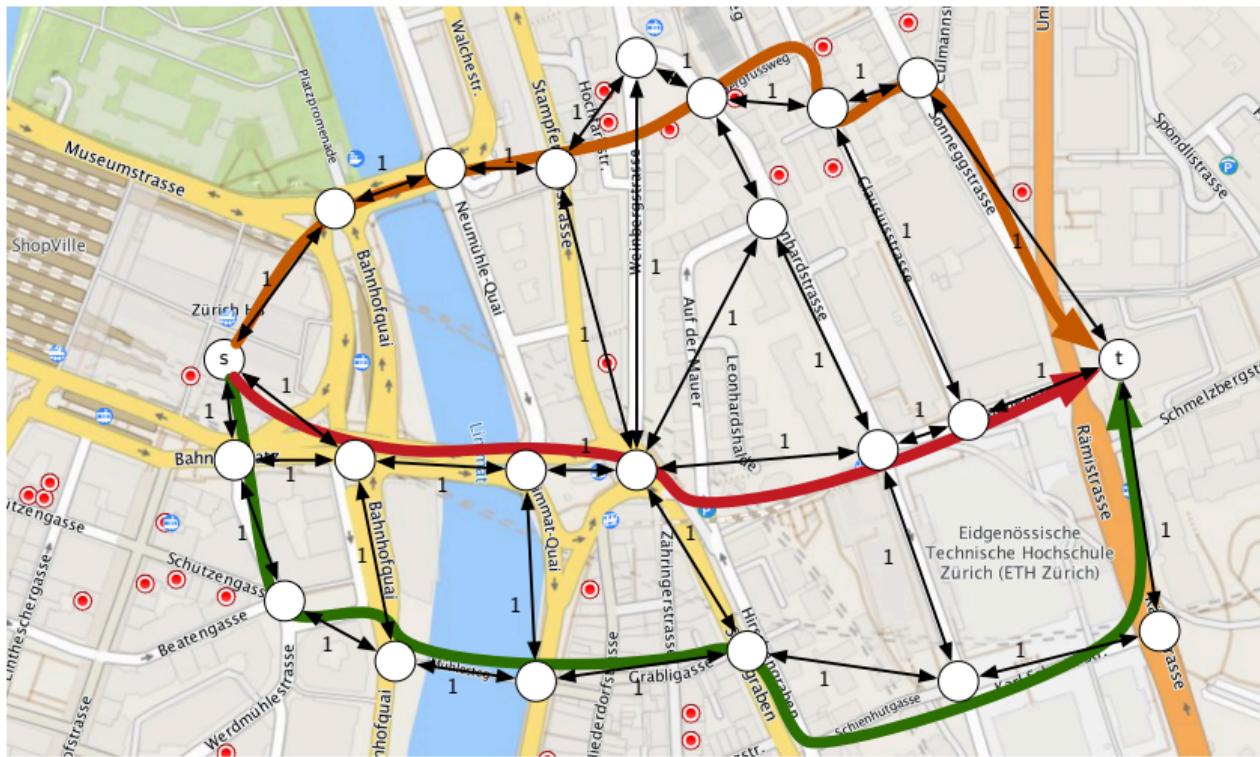
How many ways are there to get from HB to CAB without using the same street twice?

- ▶ Does this look like a flow problem? Not immediately.
- ▶ Can it be turned into a flow problem? Maybe.
- ▶ What we need according to the problem definition
 - ▶ $G = (V, E)$, directed street graph by adding edges in both directions for each street.
 - ▶ $s, t \in V$, intersections of HB and CAB.
 - ▶ $c : E \rightarrow \mathbb{N}$ with all capacities set to 1.

Lemma (Flow Decomposition)

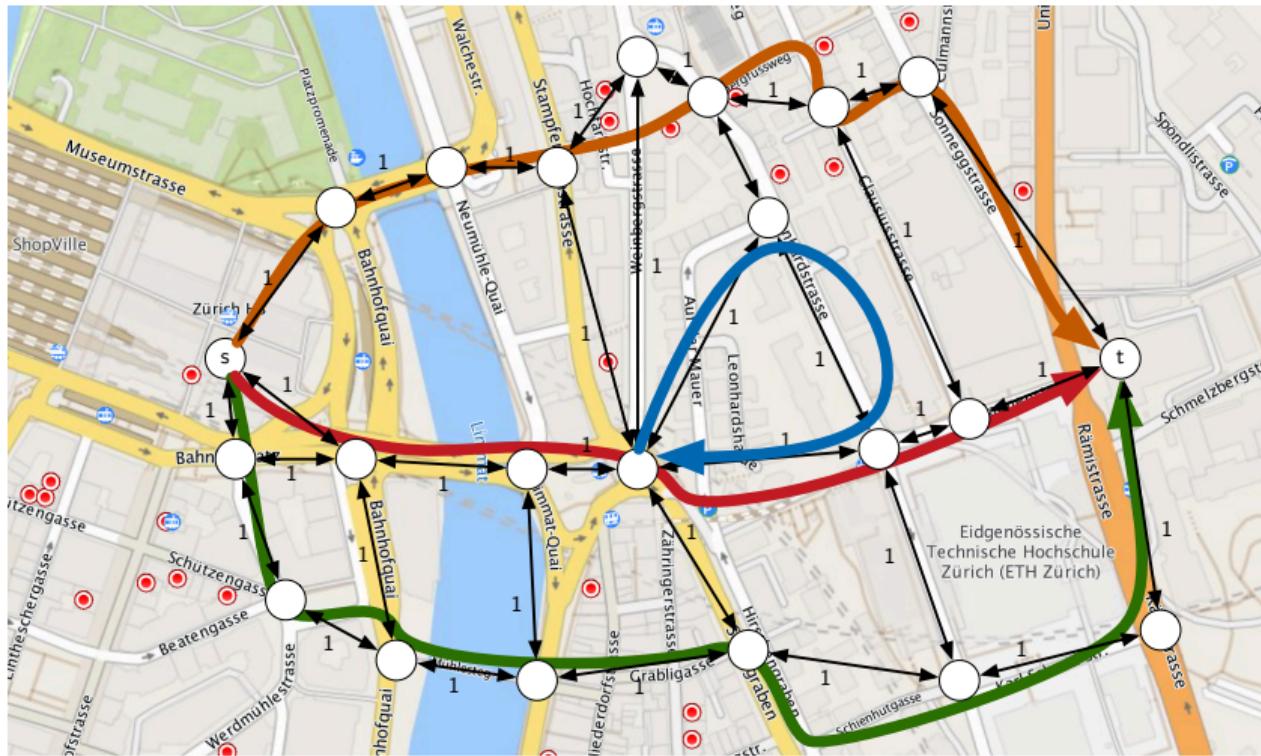
In a directed graph with unit capacities, the maximum number of edge disjoint s-t-paths is equal to the maximum flow from s to t.

Flow Application: Edge Disjoint Paths



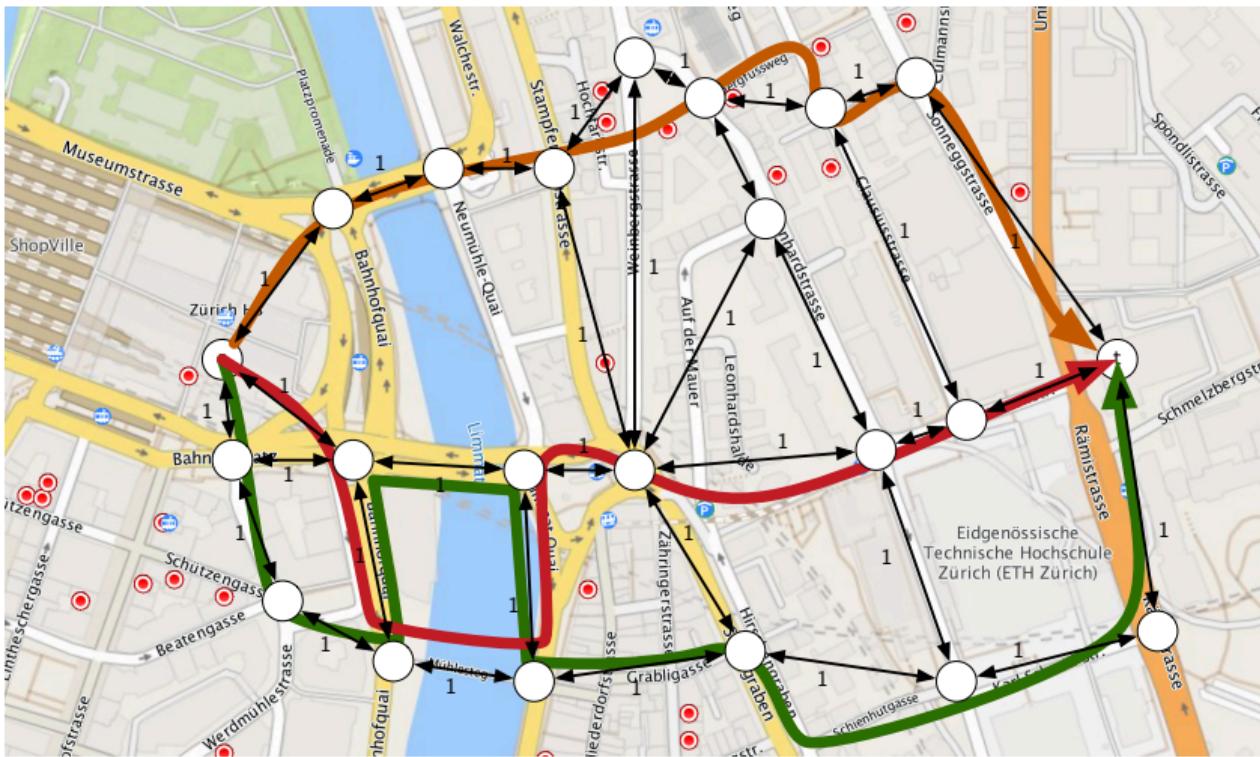
Map:
search.ch,
TomTom,
swisstopo,
OSM

Flow Application: Edge Disjoint Paths: Handling Additional Cycles



Map:
search.ch,
TomTom,
swisstopo,
OSM

Flow Application: Edge Disjoint Paths: Handling Crossing Paths



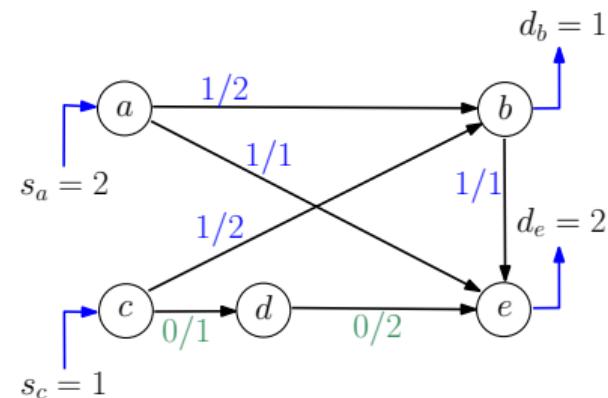
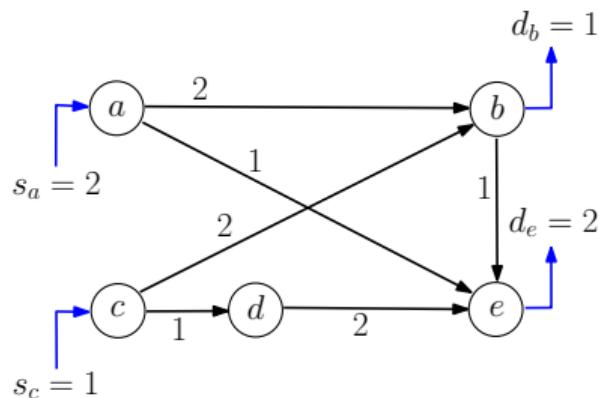
Map:
search.ch,
TomTom,
swisstopo,
OSM

Circulation Problem

- **Demand** d_v or **supply** s_v per vertex.
- Question: Is there a feasible flow $f(e)$ s. t.

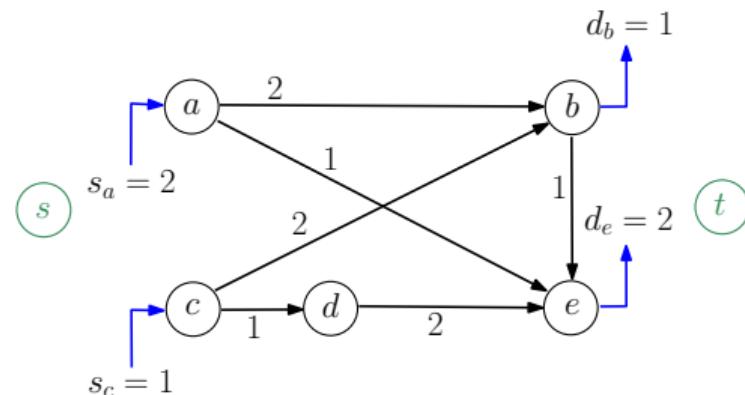
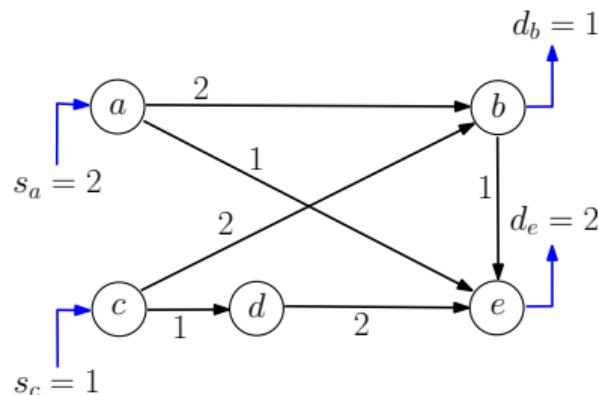
for every vertex with a demand, $d_v = \sum_{e \in \text{in-edges}(v)} f(e) - \sum_{e \in \text{out-edges}(v)} f(e)$ and

for every vertex with a supply, $s_v = \sum_{e \in \text{out-edges}(v)} f(e) - \sum_{e \in \text{in-edges}(v)} f(e)$?



Circulation Problem – solving with max-flow

- ▶ Solve using max-flow by adding source s and target t .
- ▶ Connect source to every supply ($s_v > 0$) vertex v with capacity s_v .
- ▶ Connect target to every demand ($d_v > 0$) vertex v with capacity d_v .
- ▶ Answer: If max flow equals the sum of demands, the circulation problem is solved.

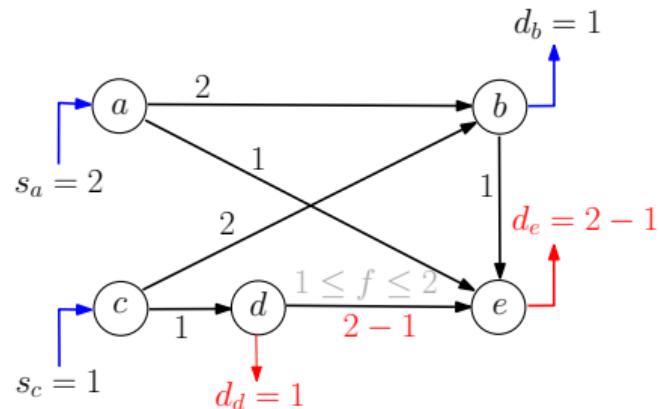


Circulation Problem – with minimum edge constraints

- ▶ Minimum edge constraint: For edge $e = (u, v)$, require

$$c_{\min}(e) \leq f(e) \leq c_{\max}(e).$$

- ▶ Increase demand of u : $d_u \leftarrow d_u + c_{\min}$.
- ▶ Increase supply of v : $s_v \leftarrow s_v + c_{\min}$.
- ▶ Decrease edge capacity of e : $c(e) \leftarrow c_{\max} - c_{\min}$.



Tutorial Problem: Soccer Prediction

- ▶ Swiss Soccer Championship, two rounds before the end.
- ▶ 2 points awarded per game, split 1-1 if game ends in a tie.

Team	Points	Remaining Games
FC St. Gallen (FCSG)	37	FCB, FCW
BSC Young Boys (YB)	36	FCW, FCB
FC Basel (FCB)	35	FCSG, YB
FC Luzern (FCL)	33	FCZ, GCZ
FC Winterthur (FCW)	31	YB, FCSG

- ▶ Can FC Luzern still win the Championship? (Assume favourable tie-breaking).
37 points still possible, so yes?
But at least one other team will have to score more, so no?
- ▶ Does this look like a flow problem? Not immediately.

Tutorial Problem: Modelling

Simplifying assumptions: FC Luzern will win all remaining games.

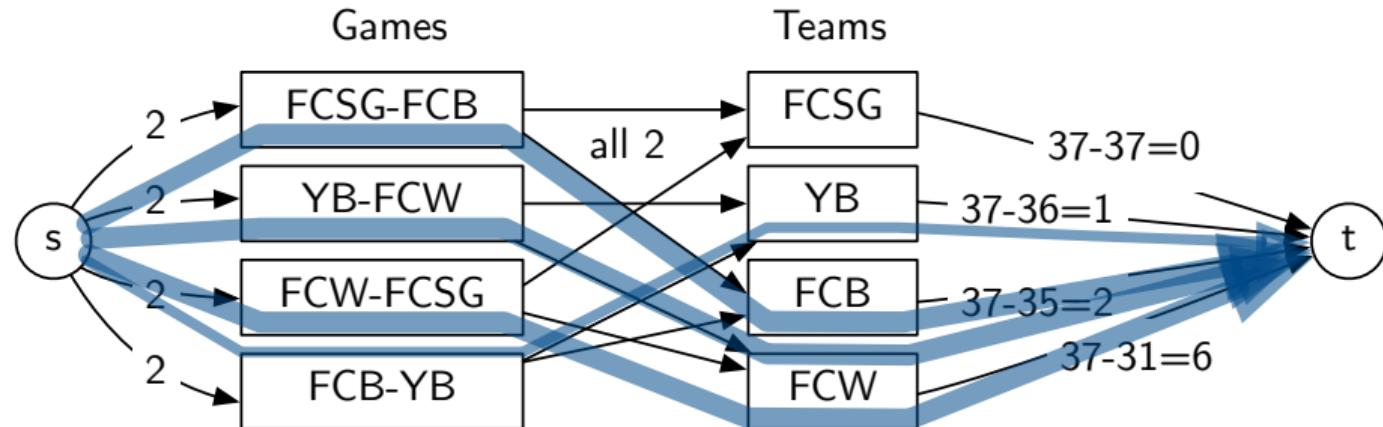
Flow problem modelling:

- ▶ graph with vertices denoting teams and games
- ▶ source, target candidates?
 - ▶ games *distribute* points
 - ▶ teams *receive* points
- ▶ edges between games and teams
- ▶ capacities:
 - ▶ 2 points to distribute per game
 - ▶ $37 - x$ limit on the points received by a team with score x

Final question: Can we let the points *flow* from the games to the teams so that all the teams end up with at most 37 points?

Tutorial Problem: Modelling

Final question: Can we let the points *flow* from the games to the teams so that all the teams end up with at most 37 points?



The answer is yes if and only if the maximum flow is $\#points = 2 \cdot \#\text{games} = 8$.

Note: This approach fails for the current system with 3:0, 1:1, 0:3 splits.

Tutorial Problem: Analysis

- ▶ Does it look like a flow problem now? Yes.
- ▶ What does a unit of flow stand for? A point in the soccer ranking.
- ▶ How large is this flow graph?
For N teams, M games, we have $n = 2 + N + M$ nodes and $m = N + 3M = \mathcal{O}(n)$ edges.
Flow is at most $2M = \mathcal{O}(n)$.
- ▶ Which algorithm should we use?
Push-Relabel runs in $\mathcal{O}(n^3)$.
Edmonds-Karp runs in $\mathcal{O}(m|f|) = \mathcal{O}(n^2)$.
Push-Relabel is still faster in practice.

Flow Problems: Summary and Outlook

Summary

What we have seen today:

- ▶ "classical" network flow problem
- ▶ graph modifications for variations
- ▶ solves edge disjoint paths
- ▶ solves circulation

What you will see in the problems:

- ▶ more techniques and combination with other algorithms like binary search

CGAL-LP

Flow-related in the previous CGAL tutorial:

- ▶ **linear programming** formulation
(equivalent, but slower
as we used a more powerful tool)

BGL-Outlook

What we will see in the next BGL tutorial:

- ▶ relation to **minimum cuts**
- ▶ extras for **bipartite graphs**: max matchings, independent sets, **min vertex cover**
- ▶ flows with **cost**: Min Cost Max Flow problem, applications and algorithms