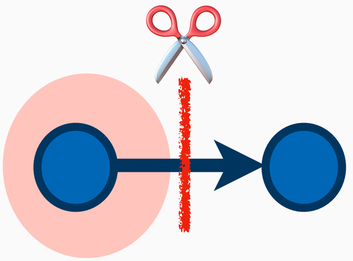
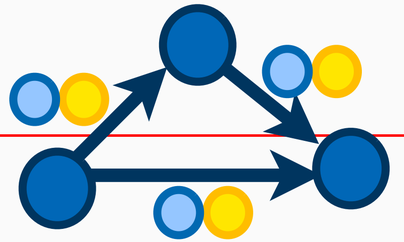
Max Flow Min Cut = MFMC



Minimum Cut, Bipartite Matching and Minimum Cost Maximum Flow with BGL



Min Cost Max Flow = MCMF

Daniel Graf (some slides by Andreas Bärtschi)

November 15, 2017

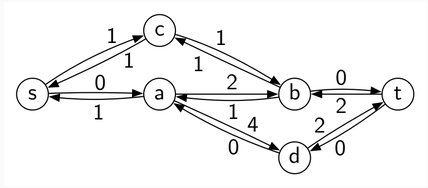
ETH Zürich

Recap: Basic Network Flows – What did we see last time?

One problem to rule them all...

Network Flow Algorithms: Ford-Fulkerson and Edmonds-Karp

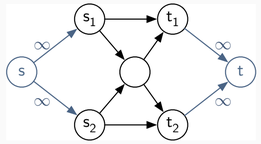
- Solution: Keep track of the flow and allow paths that *reroute* units of flow.
  These are called *augmenting paths* in the *residual network*.
- Ford-Fulkerson (1955): Repeatedly take any augmenting path: running time $\mathcal{O}(m|f|)$.
- Edmonds-Karp (1972) / Dinitz (1970): Repeatedly take any *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: $|f| \leq n \max c$



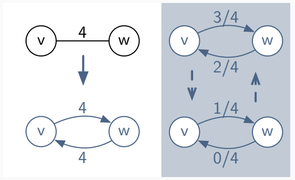
6

Common tricks

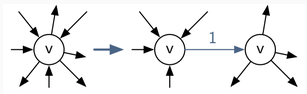
Multiple sources/sinks
with e.g. $\infty \approx \sum_{e \in E} c(e)$



Undirected graphs
antiparallel edges with flow reducable to one direction



Vertex capacities
split into in-vertex and out-vertex



Minimum flow per edge
how to enforce $c_{\min}(e) \leq f(e) \leq c_{\max}(e)$?

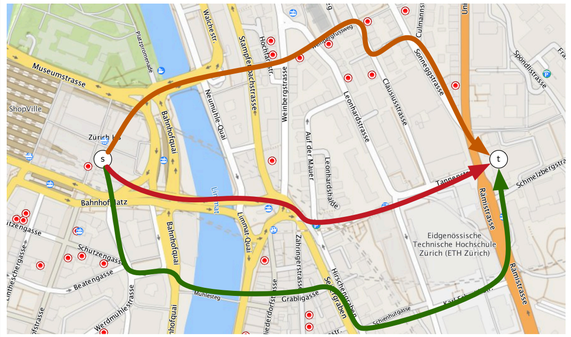
[Exercise]

21

see solution for Kingdom Defence

Flow Application: Edge Disjoint Paths

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

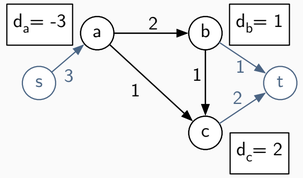


Map: search.ch, TomTom, swisstopo, OSM

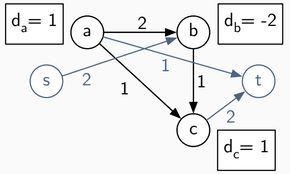
22

Flow Application: Circulation Problem

- Multiple sources with a certain amount of flow to give (**supply**).
- Multiple sinks that want a certain amount of flow (**demand**).
- Model these as negative or positive demand per vertex d_v.
- Question: Is there a feasible flow? Surely not if $\sum_{v \in V} d_v \neq 0$. Otherwise?
  Add super-source and super-sink to get a maximum flow problem.



feasible flow exists



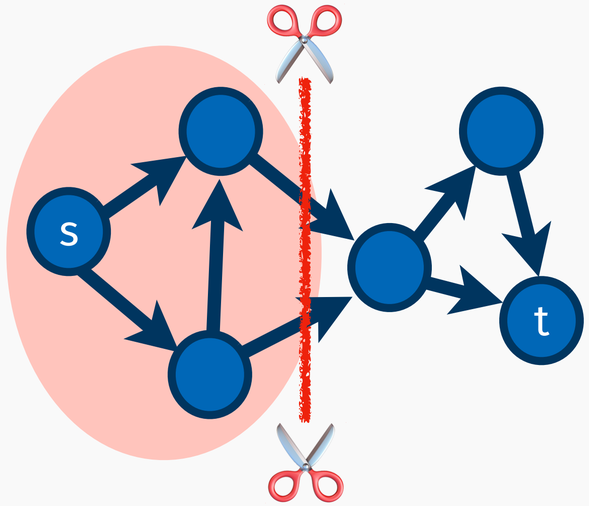
no feasible flow exists

28

Today: Advanced Network Flow – What else are flows useful for?

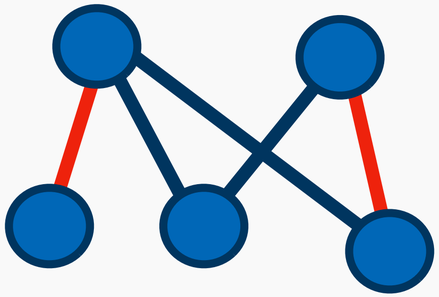
Minimum Cuts

- How to disconnect t from s cheaply?



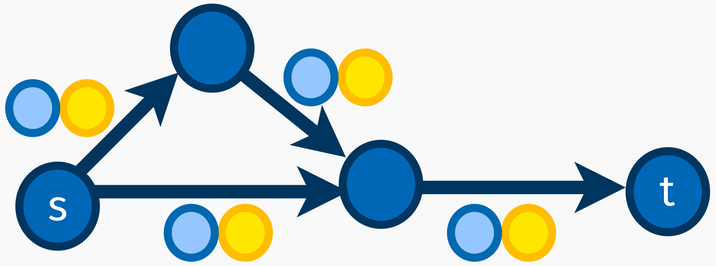
Bipartite Matching

- How to assign A's to B's effectively?



Flows with Costs

- What if sending flow comes with a price?



Minimum Cut

Minimum Cut: Shopping Trip

Minimum Cut: Shopping Trip

Start from HB:

- Visit as many shops as possible.
- Return to HB after each shop.

Condition: Use each road on at most one trip.

Compute **the bottleneck**, i.e. the number of **edge-disjoint paths**. $\Rightarrow$ Four shops.

Unrealistic condition!

(There are interesting streets in Zürich.)

Minimum Cut: Shopping Trip

Start from HB:

- Visit as many shops as possible.
- Return to HB after each shop.

Condition: **Use beautiful roads more often.**

Use route via Lindenhof at most twice.

Minimum Cut: Shopping Trip

Start from HB:

- Visit as many shops as possible.
- Return to HB after each shop.

Condition: **Use beautiful roads more often.**

Use Bahnhofstrasse up to three times.

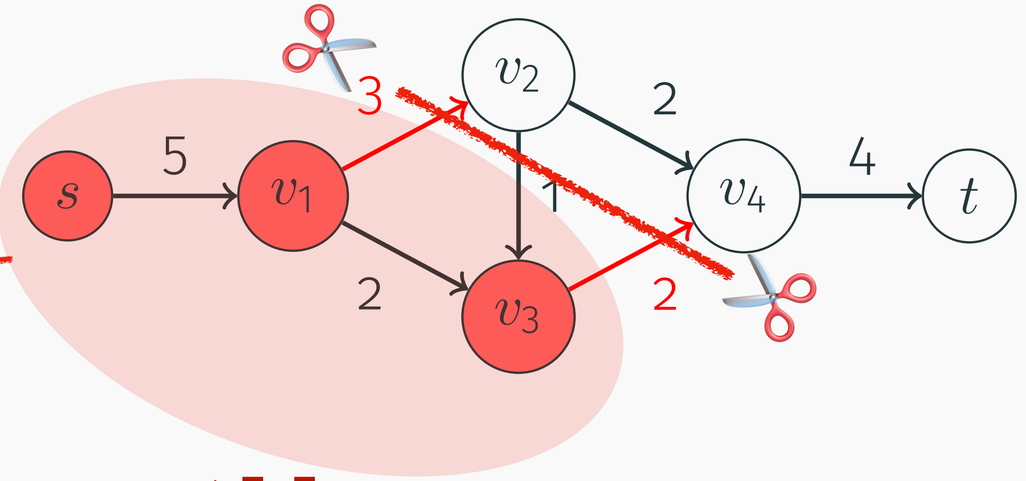
Compute **the weighted bottleneck**, i.e. the **minimum cut between s and t.** $\Rightarrow$ 6 shops.

Minimum Cut: Cuts and Flows

$G = (V, E, s, t)$ a flow network. $S \subset V$ s.t. $s \in S, t \in V \setminus S$, e.g. $\boldsymbol{S = \{s, v_1, v_3\}}$.

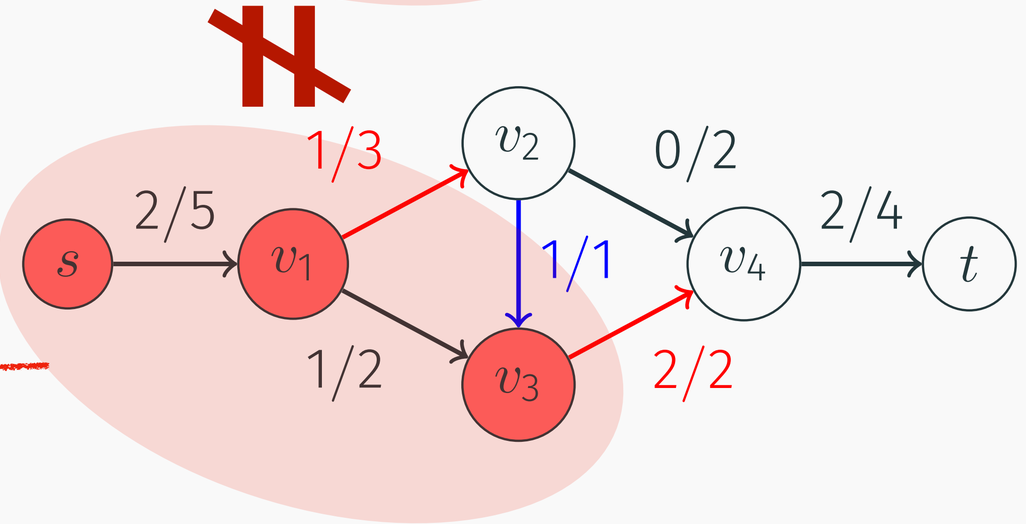
The value of the $(S, V \setminus S)$-cut is

$$\begin{aligned} \text{cap}(S, V \setminus S) &:= \text{ outgoing capacity} \\ &= \sum_{\substack{e=(u,v) \\ u \in S, v \in V \setminus S}} \text{cap}(e) \\ &= 3 + 2 = 5. \end{aligned}$$



The value of the flow f from S to $V \setminus S$ is

$$\begin{aligned} f(S, V \setminus S) &:= \text{ outgoing flow} - \text{incoming flow} \\ &= \sum_{\substack{e=(u,v) \\ u \in S, v \in V \setminus S}} \text{flow}(e) - \sum_{\substack{e=(v,u) \\ u \in S, v \in V \setminus S}} \text{flow}(e) \\ &= 1 + 2 - 1 = 2. \end{aligned}$$



Minimum Cut: Maxflow-Mincut-Theorem

Theorem (Maxflow-Mincut-Theorem)

Let f be an s-t-flow in a graph G. Then f is a maximum flow if and only if

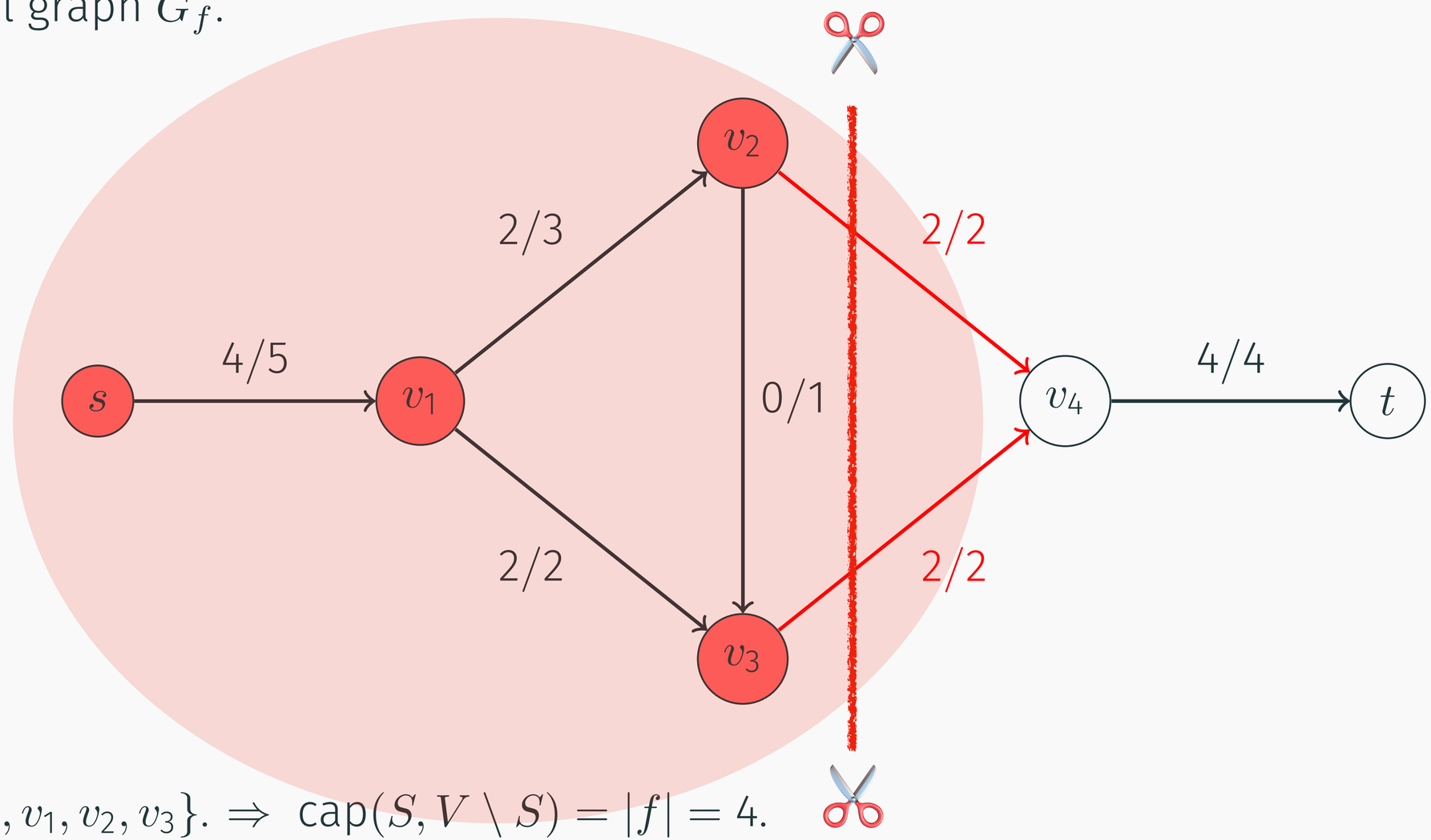
$$|f| = \min_{S:\ s \in S, t \notin S} cap(S, V \setminus S).$$

This allows us to easily find a minimum s-t-cut:

- Construct the residual graph $G_f := (V, E_f)$. For each edge $(u, v) \in G$ we have:
  - An edge $(u, v) \in G_f$ with capacity $\text{cap}(e) - f(e)$, if $\text{cap}(e) - f(e) > 0$.
  - An edge $(v, u) \in G_f$ with capacity $f(e)$, if $f(e) > 0$.
- Since f is a maximum flow, there is no s-t path in the residual graph G_f.
- Take S to be all vertices in G_f reachable from s.
  $\Rightarrow (S, V \setminus S)$ is a minimum s-t-cut.

Minimum Cut: Example

Residual graph G_f.



$|S| = \{s, v_1, v_2, v_3\}. \Rightarrow \text{ cap}(S, V \setminus S) = |f| = 4.$

Minimum Cut: Code

Example code: BFS on the residual graph G_f. → `tut9_bgl_residual_bfs.cpp`

```
90  // BFS to find vertex set S
91  std::vector<int> vis(N, false); // visited flags
92  std::queue<int> Q; // BFS queue (from std:: not boost::)
93  vis[src] = true; // Mark the source as visited
94  Q.push(src);
95  while (!Q.empty()) {
96      const int u = Q.front();
97      Q.pop();
98      OutEdgeIt ebeg, eend;
99      for (boost::tie(ebeg, eend) = boost::out_edges(u, G); ebeg != eend; ++ebeg) {
100         const int v = boost::target(*ebeg, G);
101         // Only follow edges with spare capacity
102         if (rescapacitymap[*ebeg] == 0 || vis[v]) continue;
103         vis[v] = true;
104         Q.push(v);
105     }
106 }
```

Minimum Cut: Proof of the Maxflow-Mincut-Theorem

Theorem (Maxflow-Mincut-Theorem)

Let f be an s-t-flow in a graph G. Then the following are equivalent:

1. $|f| = \min_{s \in S, t \notin S} cap(S, V \setminus S)$.
2. *f is a maxflow.*
3. *There is no s-t path in the residual graph G_f.*

Proof.

1) $\implies$ 2) f cannot be bigger than $\min_S \text{cap}(S, V \setminus S)$.

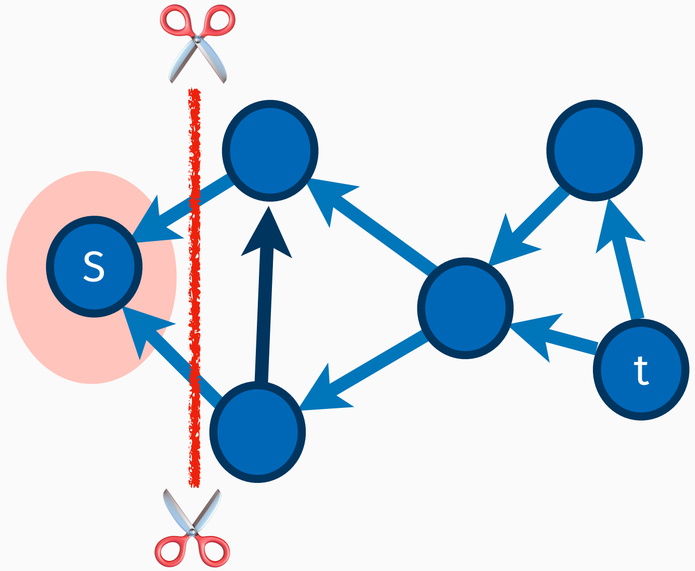
2) $\implies$ 3) Indirectly: If there was s-t path in G_f, f could be extended.

3) $\implies$ 1) Take S to be all vertices in G_f reachable from s.
Then all edges from S to $V \setminus S$ must be fully saturated by the flow.
But then $|f| = f(S, V \setminus S) = \text{cap}(S, V \setminus S)$. □

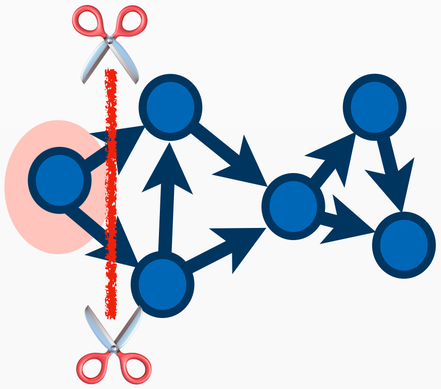
Minimum cut: Algorithm

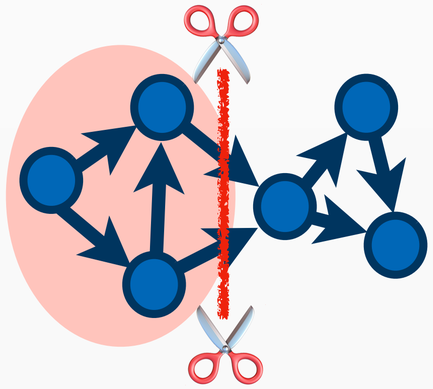
Summary of what you need to do to find a minimum cut:

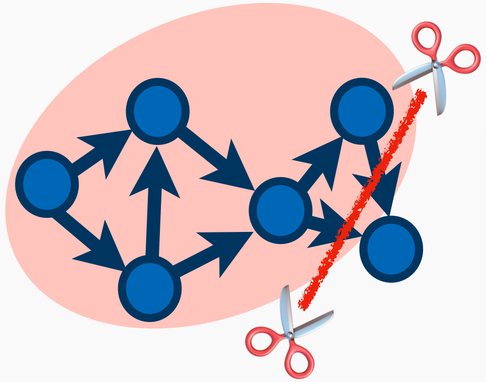
1. Compute maximum flow f and the residual graph G_f.
2. Compute the set of vertices S:
   - S is reachable from the source s in G_f.
   - BFS on edges with residual capacity > 0.
3. Output (depending on the task):
   - All vertices in S.
   - All edges going from S to $V \setminus S$.



Note: Minimum cuts are not necessarily unique. But earliest and latest min-cuts are.





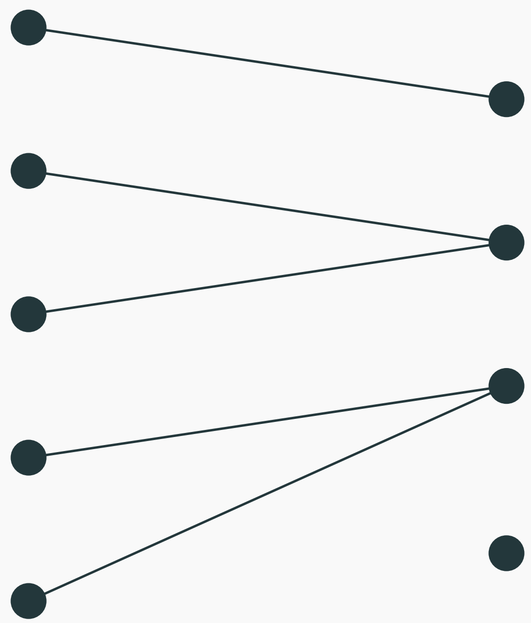


Bipartite Matchings

Maximum Matchings: Bipartite Graphs

Maximum Matching = pick as many non-adjacent edges as possible

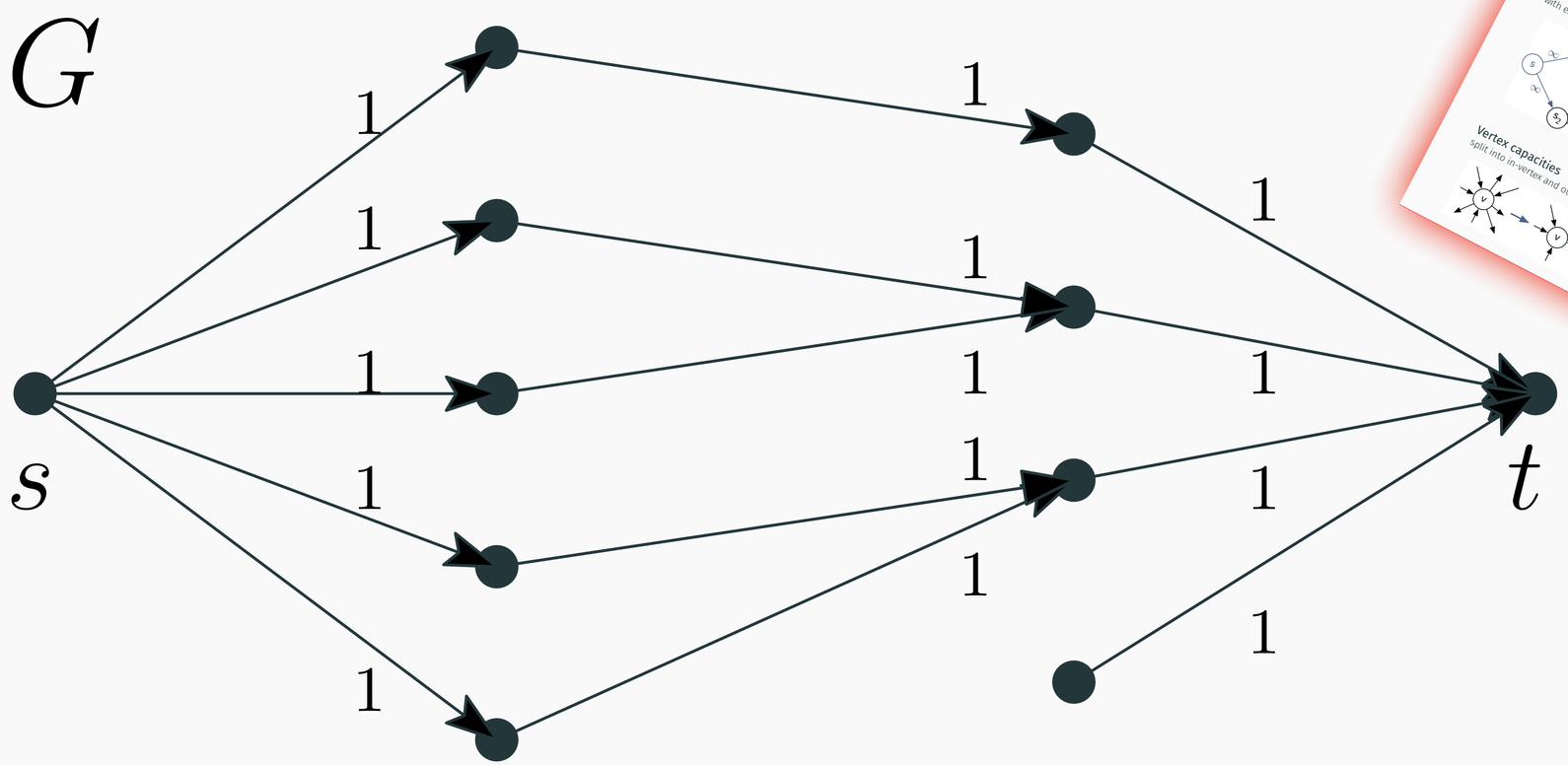
Flow formulation through circulation / vertex capacities / edges-disjoint paths:

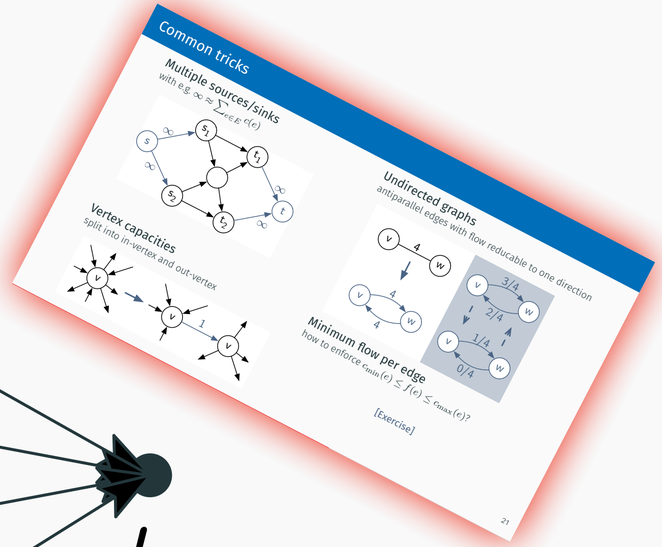


Maximum Matchings: Bipartite Graphs

Maximum Matching = pick as many non-adjacent edges as possible

Flow formulation through circulation / vertex capacities / edges-disjoint paths:

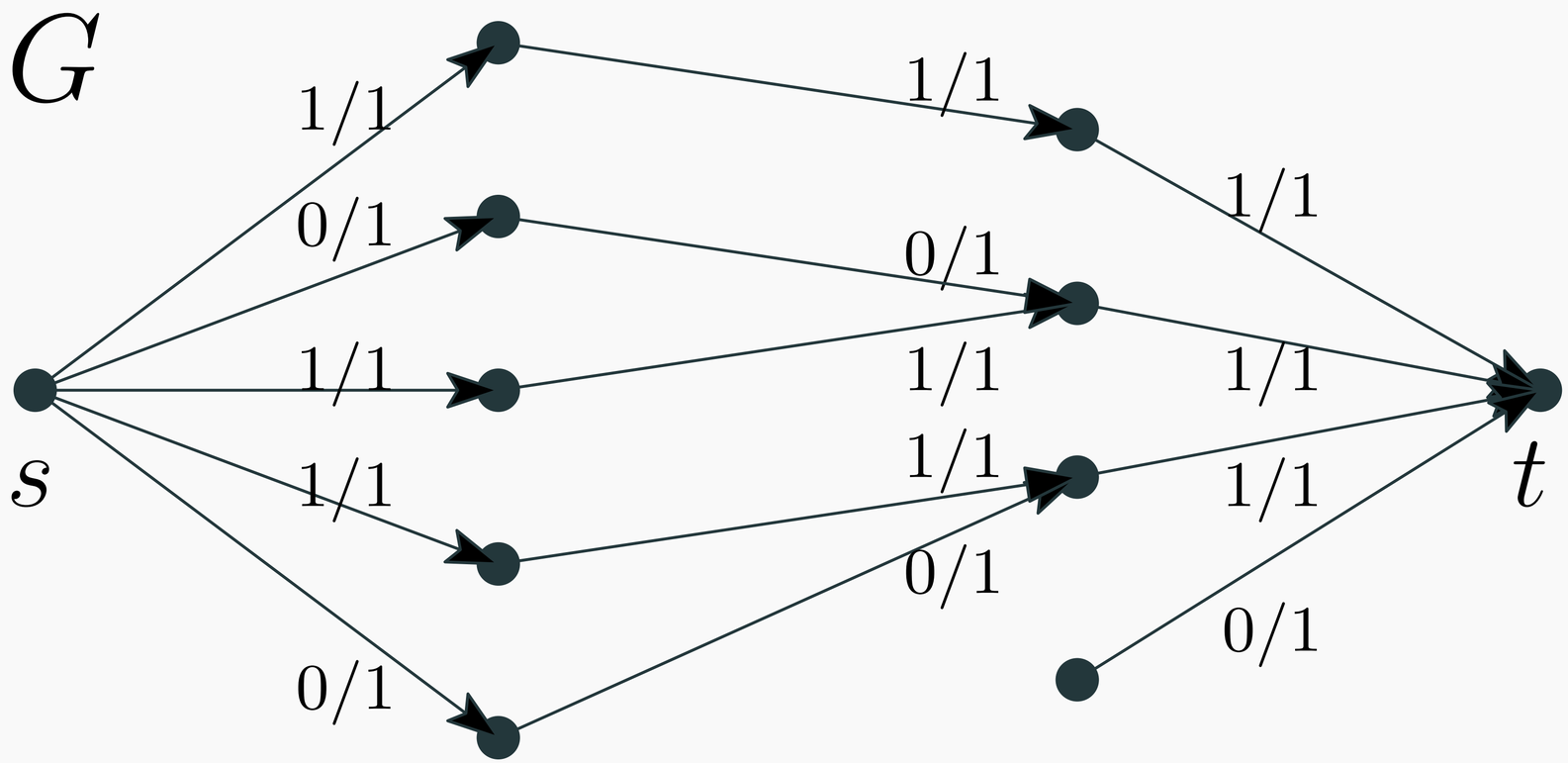




Maximum Matchings: Bipartite Graphs

Maximum Matching = pick as many non-adjacent edges as possible

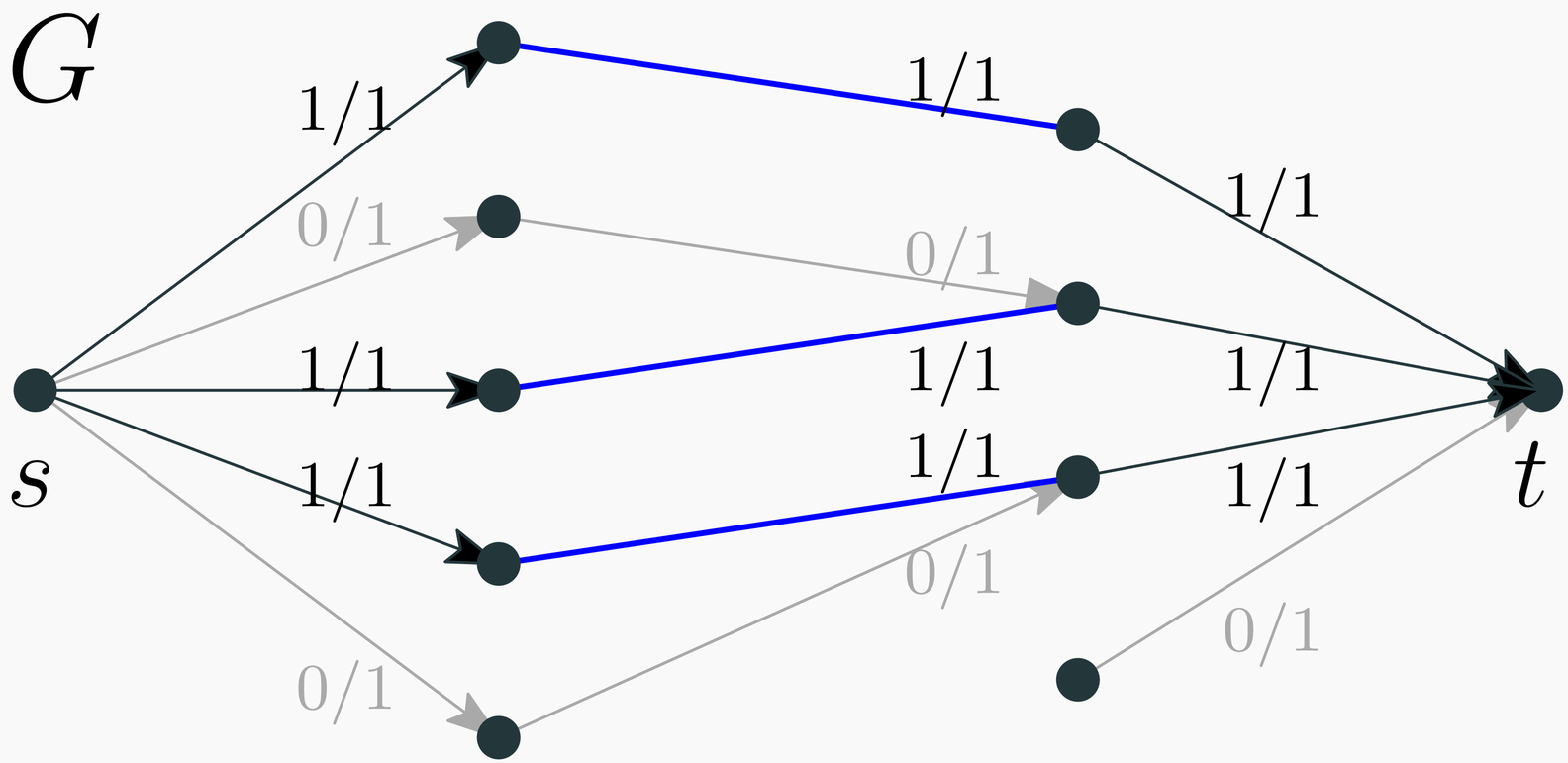
Flow formulation through circulation / vertex capacities / edges-disjoint paths:



Maximum Matchings: Bipartite Graphs

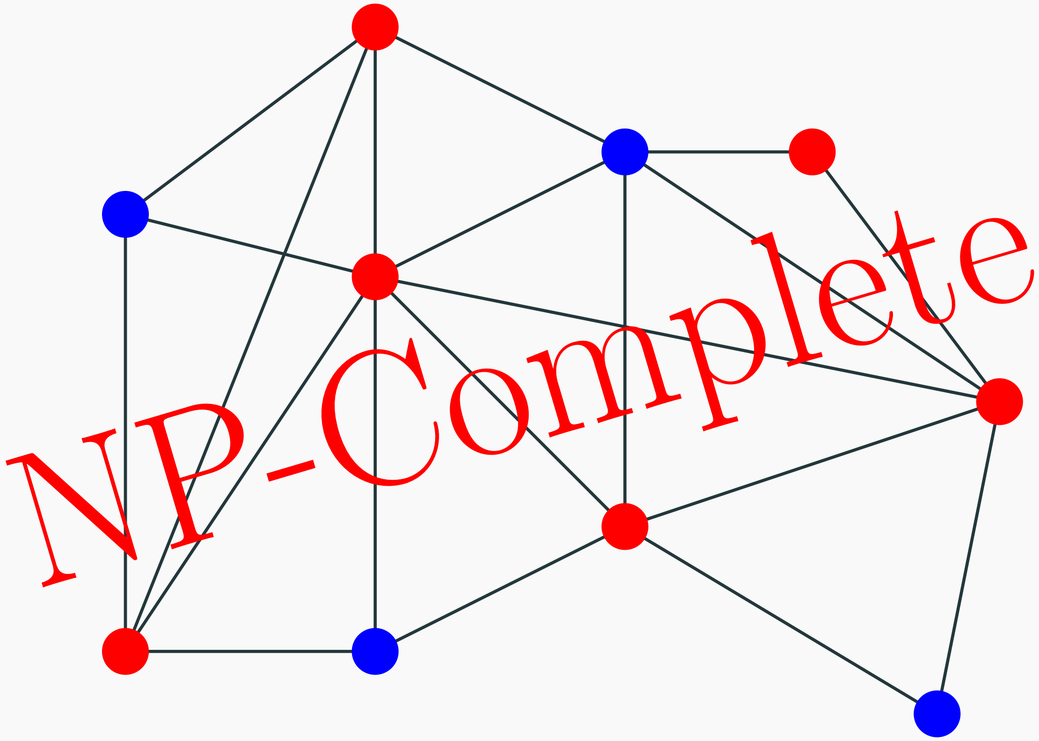
Maximum Matching = pick as many non-adjacent edges as possible

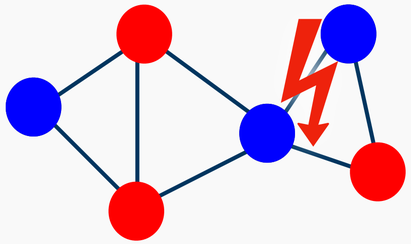
Flow formulation through circulation / vertex capacities / edges-disjoint paths:

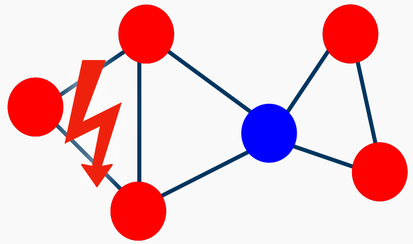


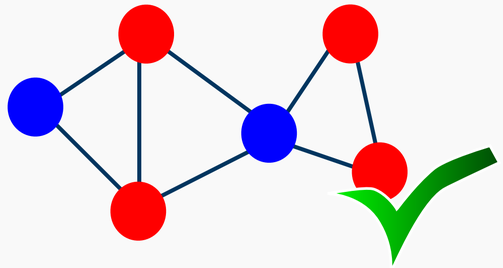
Vertex Cover and Independent Set: General Graphs

- **Maximum independent set (MaxIS)**
  Largest $T \subseteq V$, such that
  $\nexists u, v \in T : (u, v) \in E$.
- **Minimum vertex cover (MinVC)**
  Smallest $S \subseteq V$, such that
  $\forall (u, v) \in E : u \in S \vee v \in S$.
- These problems are complementary!
  MaxIS $= V \setminus$ MinVC









Brief Excursion: Options for Runtime Analysis

By now, we know many ways of deciding whether an algorithm is fast or slow:

- look at the input size (the classical way)
- look at the output size (e.g. fast as the answer is guaranteed to be small)
  If you know that MaxIS is very small, then it might be tractable while computing a MinVC directly is too slow
- look at some special input restrictions (e.g. Tracking or Jedi)
- look at detailed structure of the input (e.g. all graphs are trees)

Vertex Cover and Independent Set: Bipartite Graphs

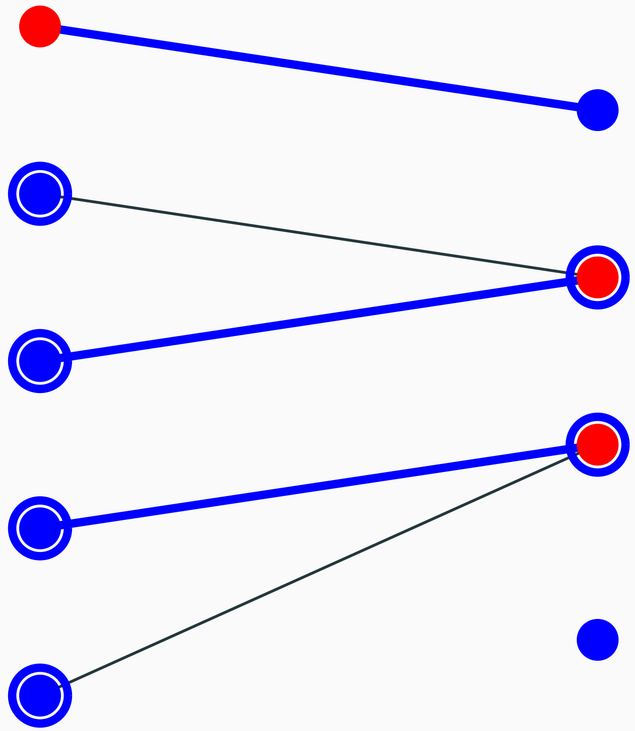
Theorem (König: MinVC and MaxIS is simpler on bipartite graphs!)

In a bipartite graph, the number of edges in a maximum matching is equal to the number of vertices in a minimum vertex cover.

Proof: See Wikipedia for a nice and short proof.

Algorithm:

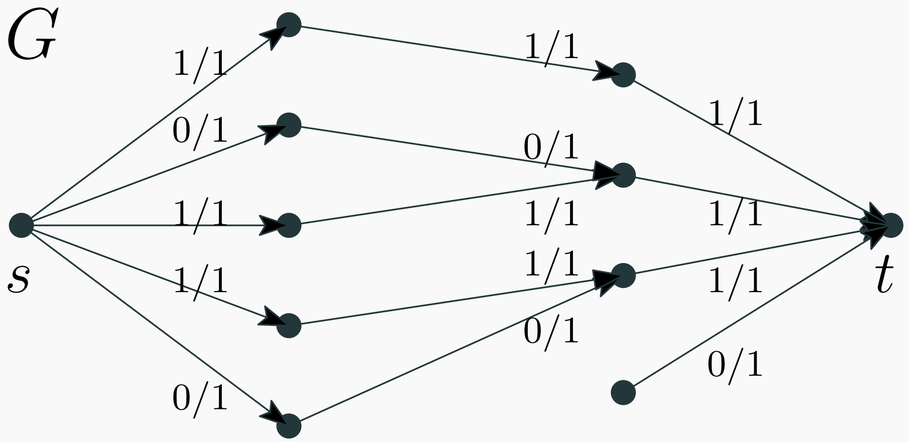
1. Maximum matching M, $V = L \cup R$. Find all unmatched vertices in L, label them as visited.
2. Starting at visited vertices search (BFS) left to right along edges from $E \setminus M$ and right to left along edges from M. Label each found vertex as visited.
3. MinVC – all unvisited in L and all visited in R.
   MaxIS – all visited in L and all unvisited in R.



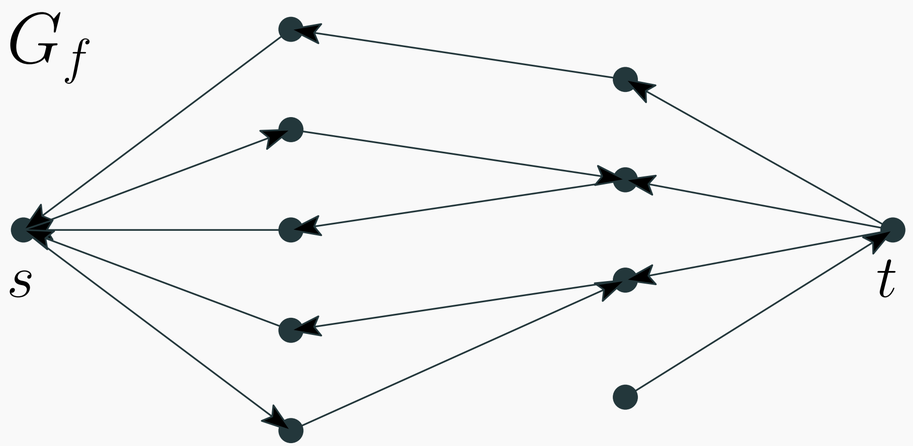
Careful! Step 2 can take several rounds.
Easy Implementation?

Finding a MinVC or MaxIS in bipartite graphs: step by step

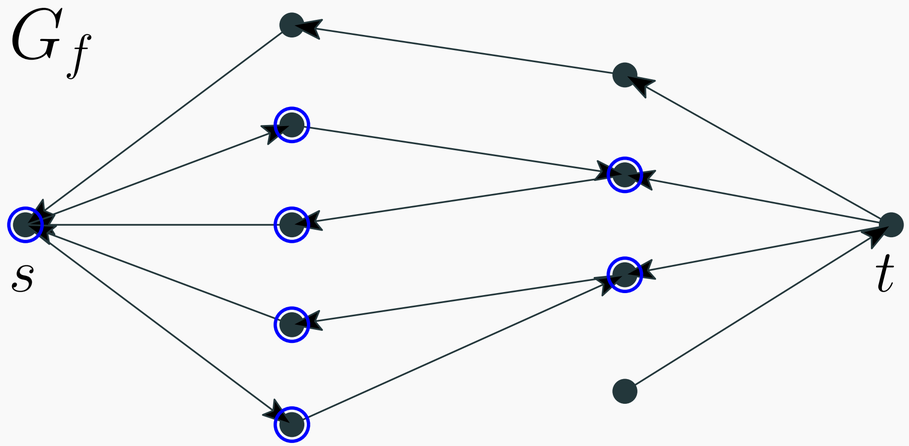
1) Formulate and compute the flow:



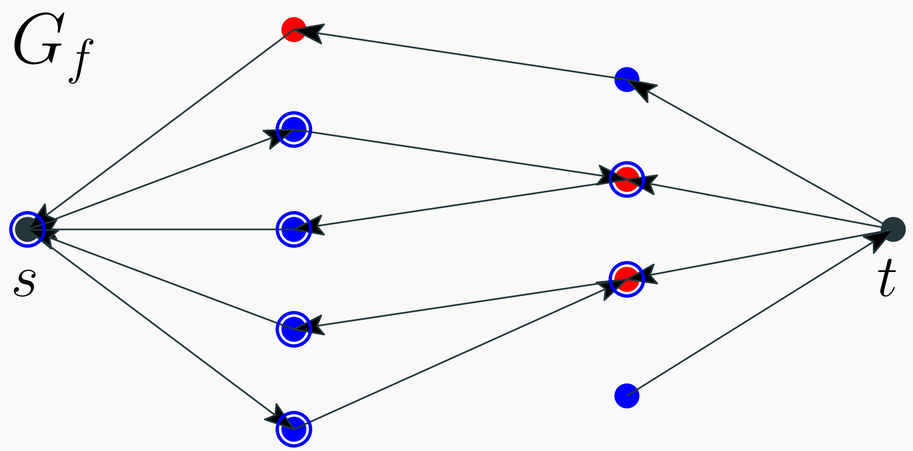
2) Compute the residual graph G_f:



3) Mark reachable vertices from s with BFS:



4) Read the MinVC or MaxIS from the marks:



Summary: MaxFlowMinCut and Bipartite Matching

What you should remember:

Edge Cut

- Theorem: maximum amount of any s-t-flow = minimum capacity of any s-t-cut
- Finding the cut: BFS/DFS on residual graph starting from s.

Vertex Cover

- Minimum vertex cover and maximum independent set are hard problems.
- Bipartite graphs allow fast MinVC and MaxIS (both on top of maximum matching).
- Finding the minimum vertex cover: BFS/DFS on residual graph from s.

Min Cost Max Flow

Minimum Cost for a Bipartite Matching

How to pick the *best* maximum matching? How to break ties among equally large ones?

E.g.: Which set of marriages is the most stable?

Or: Which consumer/producer pairing is the most effective?

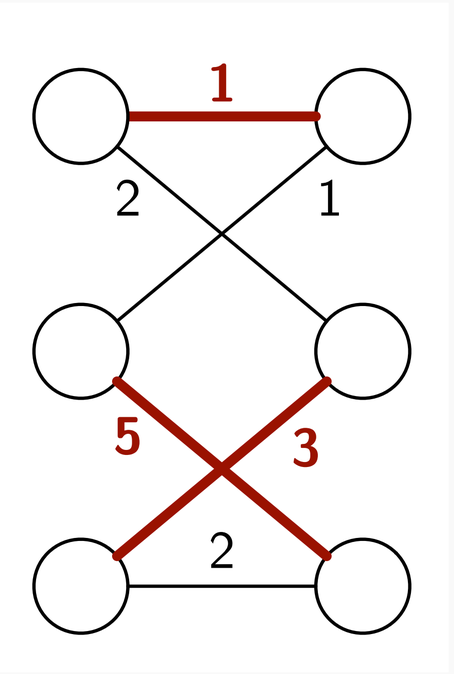
What if the edges in a matching also have costs associated?

- cardinality of the matching is no longer the only objective
- second priority:
  minimize the total cost

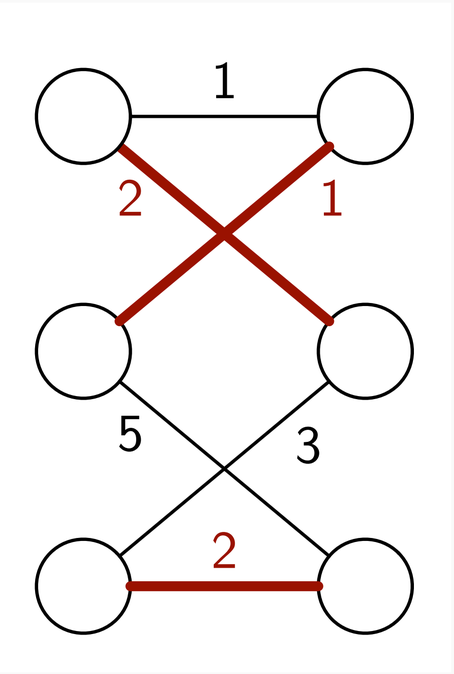
We search for the cheapest among all maximum matchings.

This does not fit into our model of flows.

two maximum matchings of different cost:



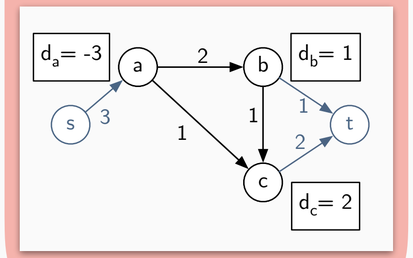
$1 + 5 + 3 = 9$



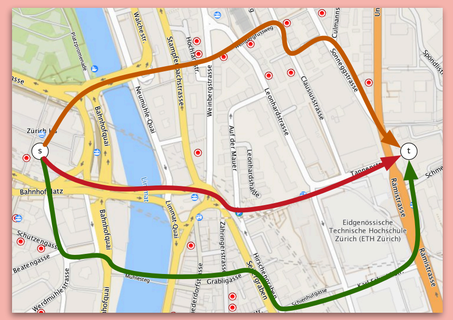
$2 + 1 + 2 = 5$

Complexity Landscape

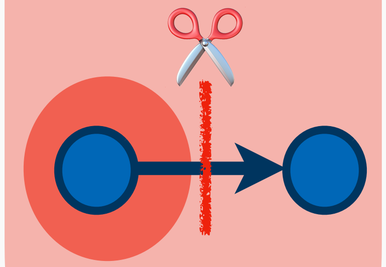
Circulation



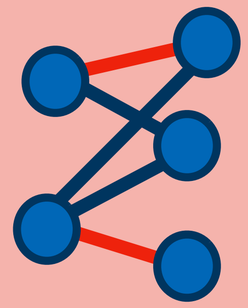
Edge-disjoint Paths



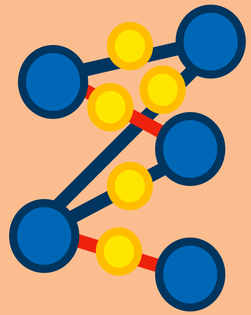
Minimum Cut

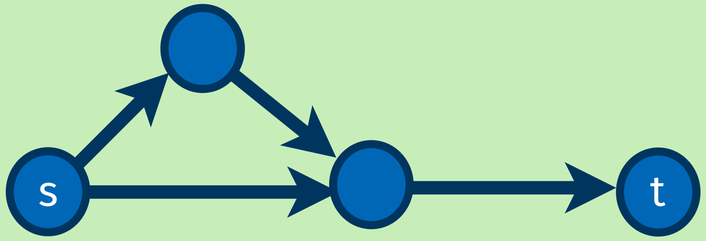


Bipatite Matching

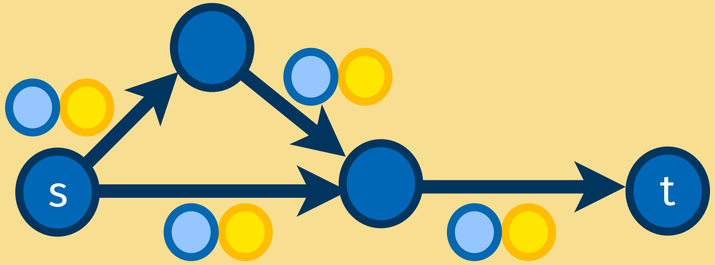


Bipartite Cost Matching





Maximum Flow



Min Cost Max Flow

More General Model: Minimum Cost Maximum Flow

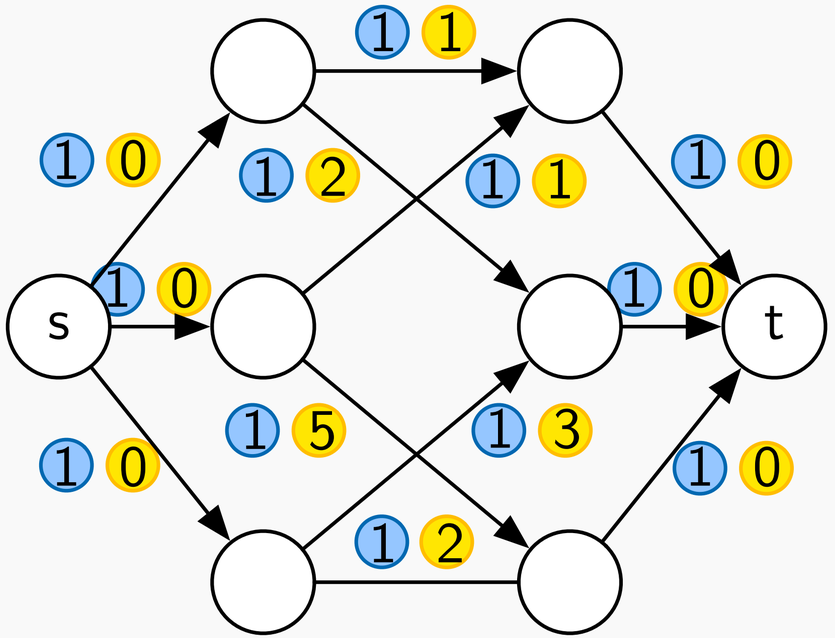
We extend the network flow problem by allowing edge costs that occur per unit of flow.

Input: A flow network consisting of

- directed graph $G = (V, E)$
- source and sink $s, t \in V$
- edge capacity $cap : E \to \mathbb{N}$
- **edge cost** $cost : E \to \mathbb{Z}$.

Output: A flow function f with minimal $cost(f) = \sum_{e \in E} f(e) \cdot cost(e)$ among all flows with maximal $|f|$.

This can model much more than just minimum cost bipartite matching.



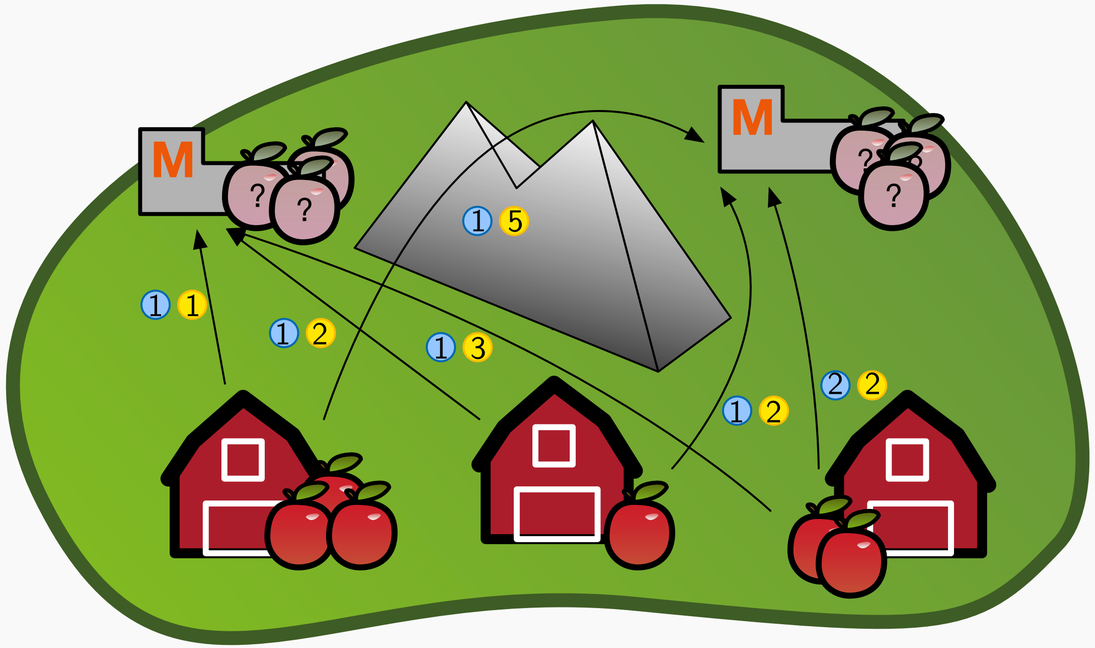
Legend: capacity vs cost

Example: Fruit Delivery

Migros wants to schedule fruit deliveries from their farmers to their shops.

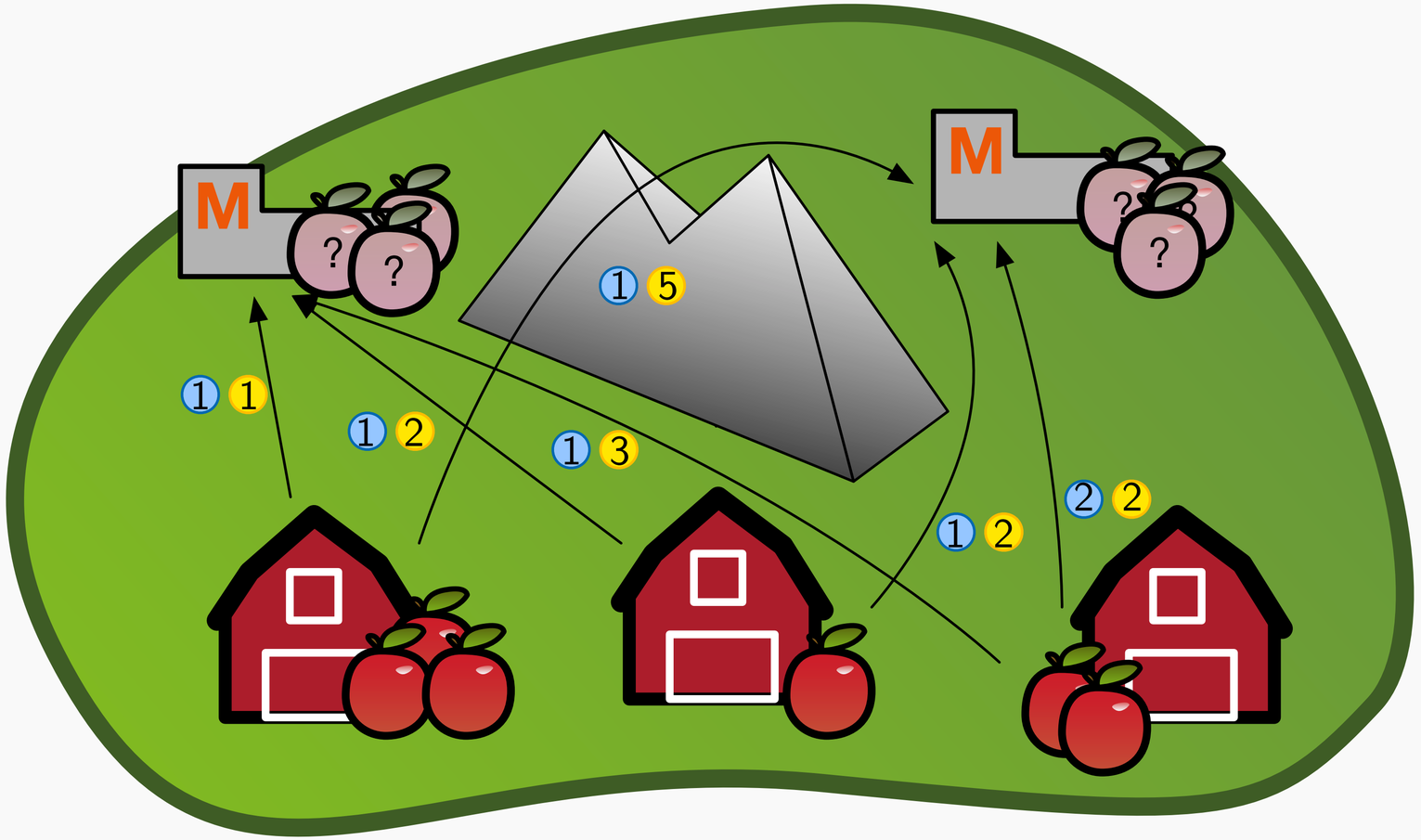
They know all important parameters;

- production per farm [in kg]
- demand per shop [in kg]
- transportation capacity [in kg] and transportation cost [in Fr. pro kg] for every farm-shop pair



Note: This is not just a bipartite matching, even though the graph is bipartite.
One farm might deliver to multiple shops (and vice versa).

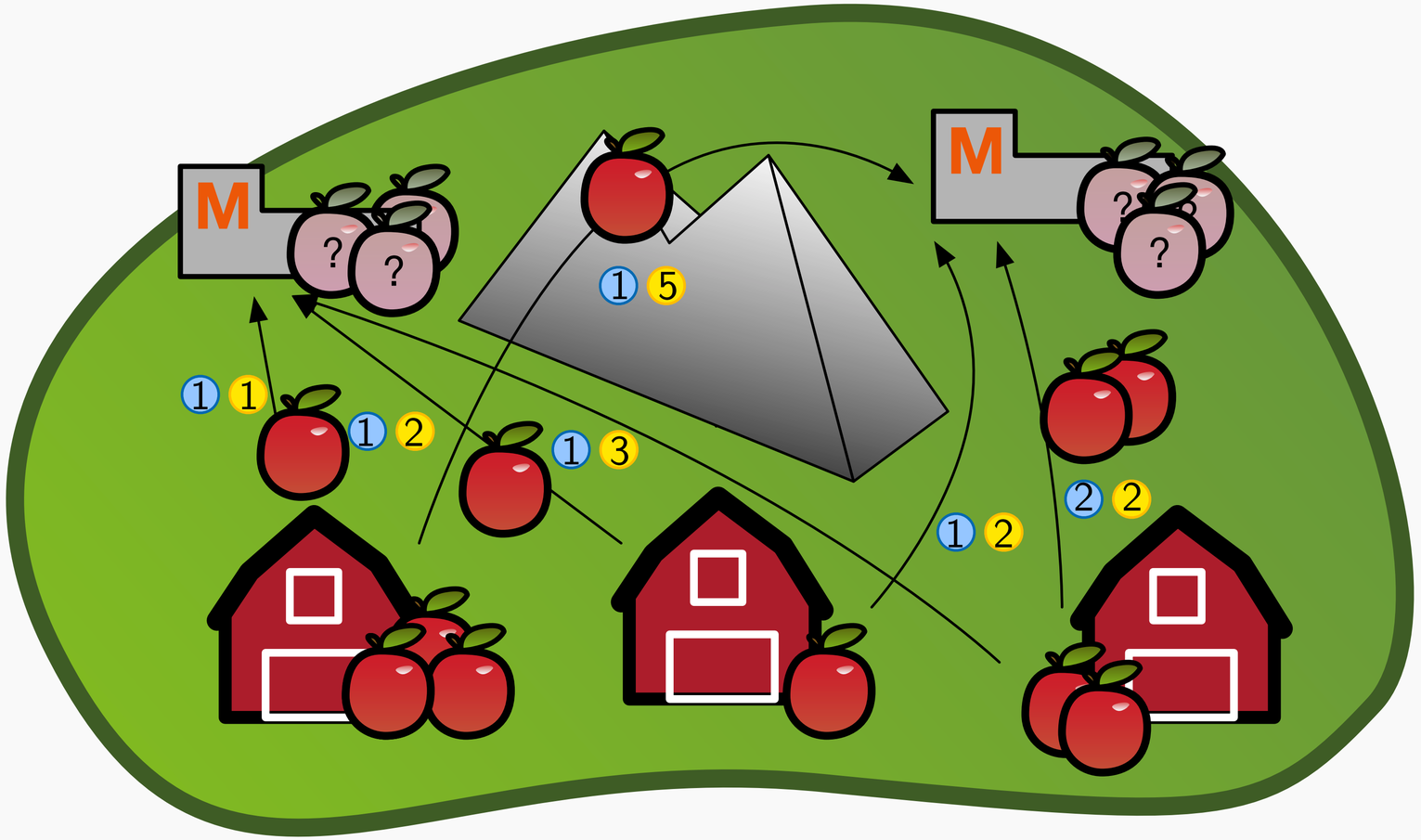
Example: Fruit Delivery



Flow: $2 + 1 + 2 = 5$

Cost: $1 \cdot 1 + 1 \cdot 5 + 1 \cdot 2 + 2 \cdot 2 = 12$

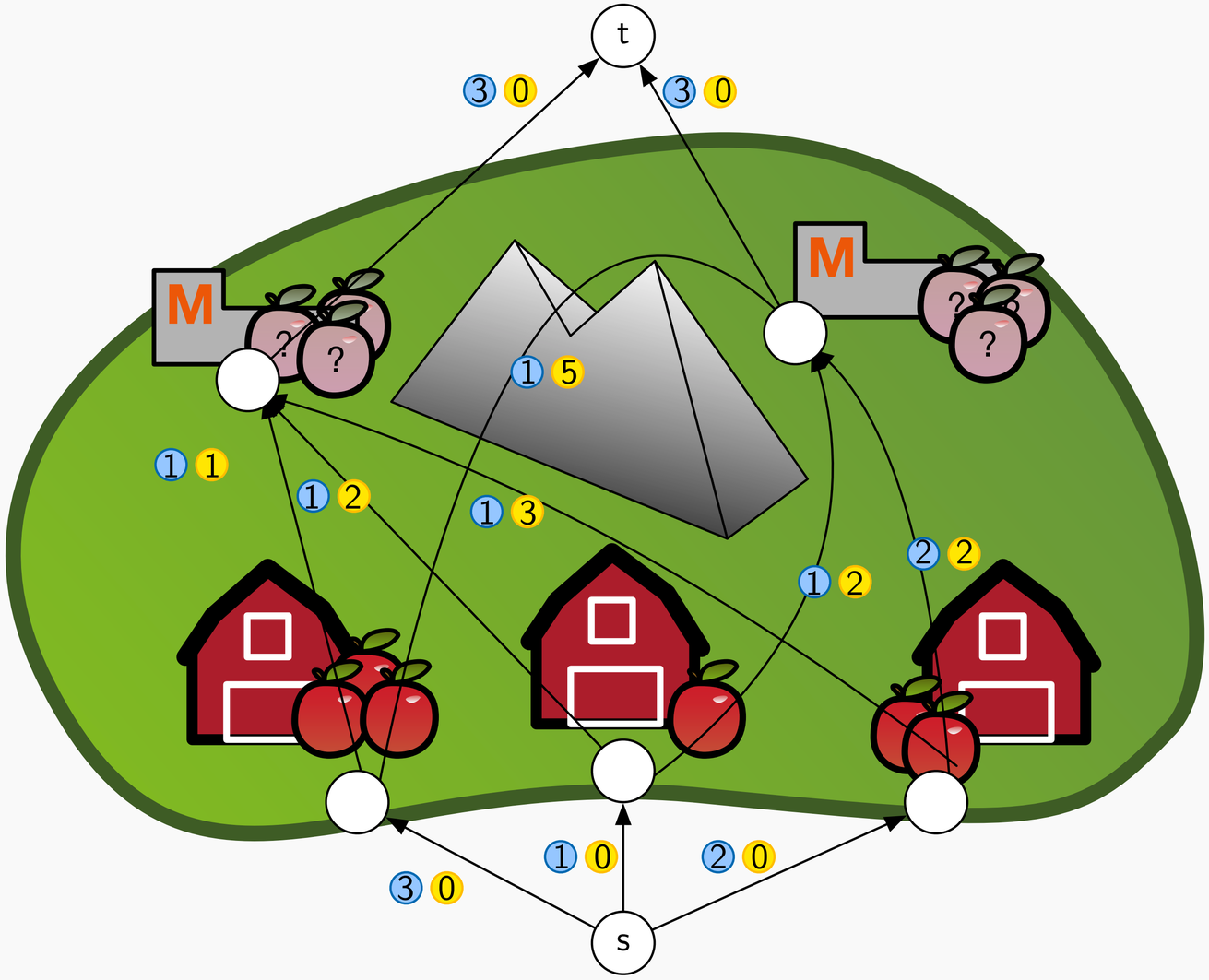
Example: Fruit Delivery



Flow: $2 + 1 + 2 = 5$

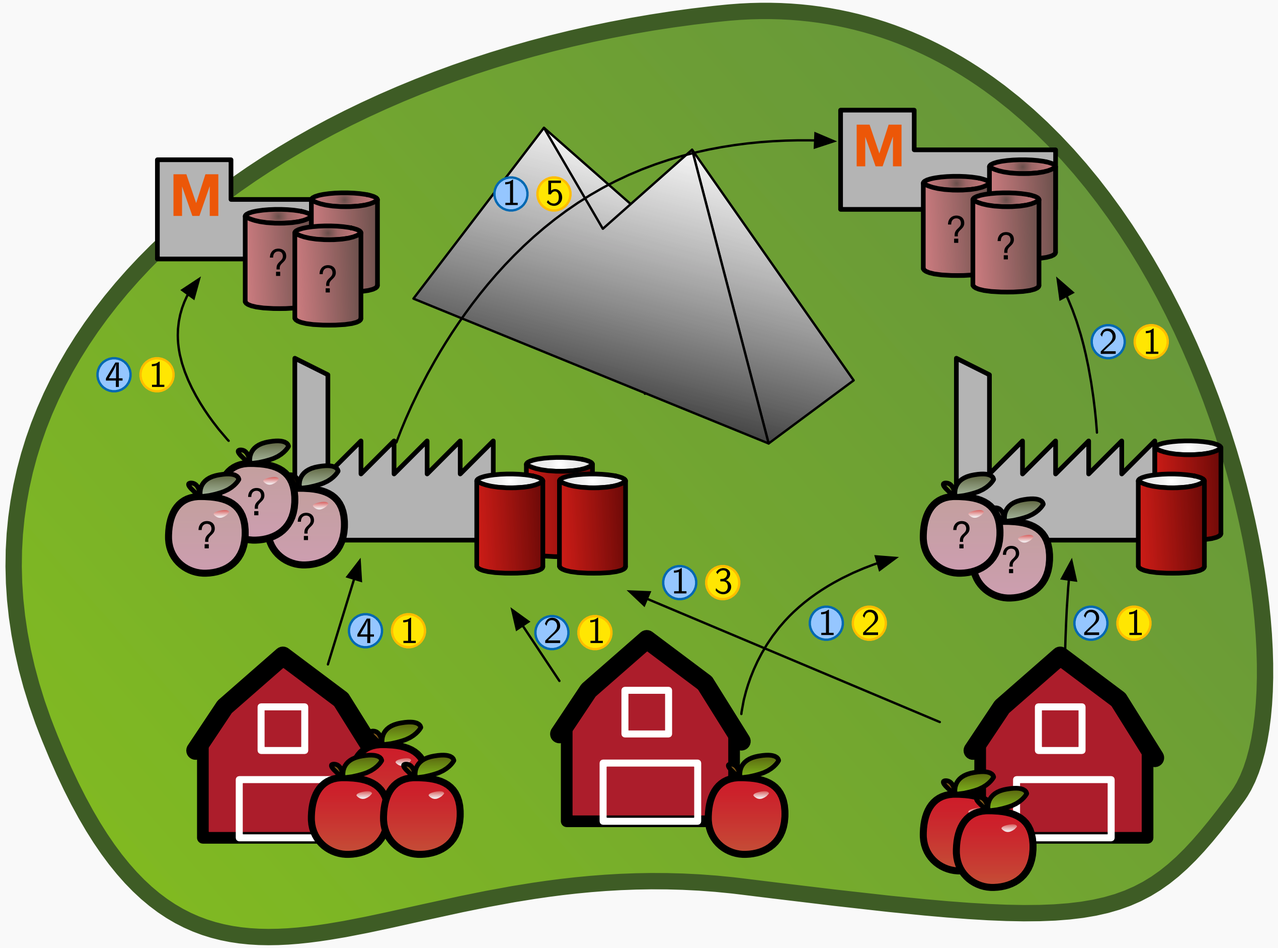
Cost: $1 \cdot 1 + 1 \cdot 5 + 1 \cdot 2 + 2 \cdot 2 = 12$

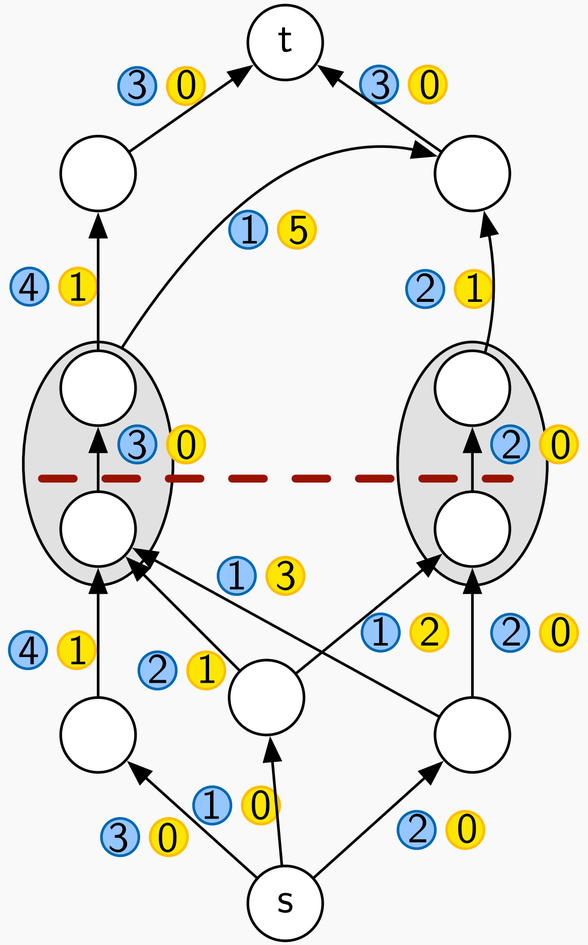
Example: Fruit Delivery



Extended Example: Canned Fruit Delivery

Extension: Canned fruit requires transportation to and from a canning factory.





Min Cost Max Flow with BGL

There are two algorithms available in BGL (available in BGL v1.55+):

- `cycle_canceling()`
  - slow, but can handle negative costs
  - needs a maximum flow to start with (call e.g. `push_relabel_max_flow` before)
  - runtime $\mathcal{O}(C \cdot (nm))$ where C is the cost of the initial flow
  - [BGL documentation], [BGL example].
- `successive_shortest_path_nonnegative_weights()`
  - faster, but works only for non-negative costs
  - sum up all residual capacities at the source to get the flow value
  - runtime $\mathcal{O}(|f| \cdot (m + n \log n))$
  - [BGL documentation], [BGL example].

Useful things to know:

- costs implemented as `edge_weight_t` property
- call `find_flow_cost()` to compute the cost of the flow

Min Cost Max Flow with BGL

Weights and capacities, just one more nesting level in the typedefs:

```
24 // Graph Type with nested interior edge properties for Cost Flow Algorithms
25 typedef boost::adjacency_list_traits<boost::vecS, boost::vecS, boost::directedS> Traits;
26 typedef boost::adjacency_list<boost::vecS, boost::vecS, boost::directedS, boost::no_property,
27     boost::property<boost::edge_capacity_t, long,
28         boost::property<boost::edge_residual_capacity_t, long,
29             boost::property<boost::edge_reverse_t, Traits::edge_descriptor,
30                 boost::property <boost::edge_weight_t, long> > > > > Graph; // new!
31 // Interior Property Maps
32 typedef boost::property_map<Graph, boost::edge_capacity_t>::type      EdgeCapacityMap;
33 typedef boost::property_map<Graph, boost::edge_weight_t >::type       EdgeWeightMap; // new!
34 typedef boost::property_map<Graph, boost::edge_residual_capacity_t>::type ResidualCapacityMap;
35 typedef boost::property_map<Graph, boost::edge_reverse_t>::type       ReverseEdgeMap;
36 typedef boost::graph_traits<Graph>::vertex_descriptor          Vertex;
37 typedef boost::graph_traits<Graph>::edge_descriptor            Edge;
38 typedef boost::graph_traits<Graph>::out_edge_iterator  OutEdgeIt; // Iterator
```

Code file: → `tut9_bgl_mincostmaxflow.cpp`

Min Cost Max Flow with BGL

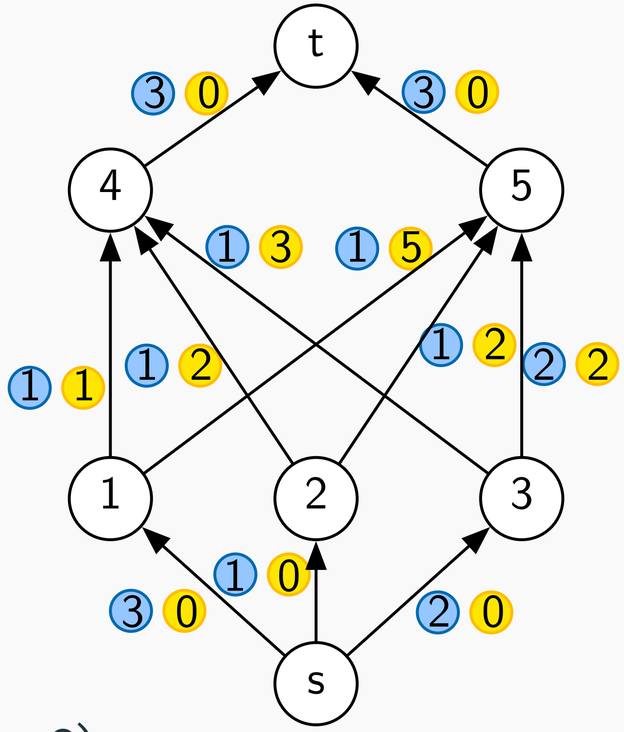
Extending the edge adder:

```
49 class EdgeAdder {
50     EdgeAdder(Graph &G, EdgeCapacityMap &capacitymap,
51         EdgeWeightMap &weightmap, ReverseEdgeMap &revedgemap)
52         : G(G), capacitymap(capacitymap), weightmap(weightmap),
53           revedgemap(revedgemap) {}
54     void addEdge(int u, int v, long c, long w) {
55         Edge e, rev_e;
56         boost::tie(e, boost::tuples::ignore) = boost::add_edge(u, v, G);
57         boost::tie(rev_e, boost::tuples::ignore) = boost::add_edge(v, u, G);
58         capacitymap[e] = c;
59         weightmap[e] = w; // new!
60         capacitymap[rev_e] = 0;
61         weightmap[rev_e] = -w; // new
62         revedgemap[e] = rev_e;
63         revedgemap[rev_e] = e;
64     }
65 };
```

Min Cost Max Flow with BGL

Building the graph

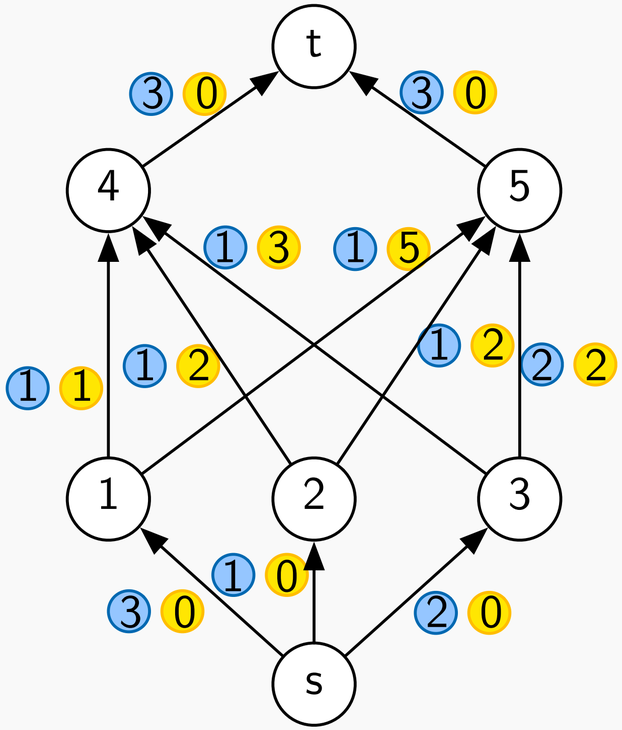
```
67 const int N=7;
68 const int v_source = 0;
69 const int v_farm1 = 1;
70 const int v_farm2 = 2;
71 const int v_farm3 = 3;
72 const int v_shop1 = 4;
73 const int v_shop2 = 5;
74 const int v_target = 6;
75
76 // Create Graph and Maps
77 Graph G(N);
78 EdgeCapacityMap capacitymap = boost::get(boost::edge_capacity, G);
79 EdgeWeightMap weightmap = boost::get(boost::edge_weight, G);
80 ReverseEdgeMap revedgemap = boost::get(boost::edge_reverse, G);
81 ResidualCapacityMap rescapacitymap = boost::get(boost::edge_residual_capacity, G);
82 EdgeAdder eaG(G, capacitymap, weightmap, revedgemap);
```



Min Cost Max Flow with BGL

Add the edges:

```
85 eaG.addEdge(v_source, v_farm1, 3, 0);
86 eaG.addEdge(v_source, v_farm2, 1, 0);
87 eaG.addEdge(v_source, v_farm3, 2, 0);
88
89 eaG.addEdge(v_farm1, v_shop1, 1, 1);
90 eaG.addEdge(v_farm1, v_shop2, 1, 5);
91 eaG.addEdge(v_farm2, v_shop1, 1, 2);
92 eaG.addEdge(v_farm2, v_shop2, 1, 2);
93 eaG.addEdge(v_farm3, v_shop1, 1, 3);
94 eaG.addEdge(v_farm3, v_shop2, 2, 2);
95
96 eaG.addEdge(v_shop1, v_target, 3, 0);
97 eaG.addEdge(v_shop2, v_target, 3, 0);
```



Min Cost Max Flow with BGL

Running the algorithm:

```
101 // Option 1: Min Cost Max Flow with cycle_canceling
102 int flow = boost::push_relabel_max_flow(G, v_source, v_target);
103 boost::cycle_canceling(G);
104 int cost = boost::find_flow_cost(G);
105 std::cout << "flow " << flow << std::endl; // 5
106 std::cout << "cost " << cost << std::endl; // 12
```

Min Cost Max Flow with BGL

Running the algorithm:

```
110 // Option 2: Min Cost Max Flow with successive_shortest_path_nonnegative_weights
111 boost::successive_shortest_path_nonnegative_weights(G, v_source, v_target);
112 int cost = boost::find_flow_cost(G);
113 // Iterate over all edges leaving the source to sum up the flow values.
114 int s_flow = 0;
115 OutEdgeIt e, eend;
116 for(boost::tie(e, eend) = boost::out_edges(boost::vertex(v_source,G), G); e != eend; ++e) {
117     s_flow += capacitymap[*e] - rescapacitymap[*e];
118 }
119 // Or equivalently, you can do the summation at the sink.
120 int t_flow = 0;
121 for(boost::tie(e, eend) = boost::out_edges(boost::vertex(v_target,G), G); e != eend; ++e) {
122     t_flow += rescapacitymap[*e] - capacitymap[*e]; // reverse for residual edges
123 }
124 std::cout << "s_out␣flow␣" << s_flow << std::endl; // 5
125 std::cout << "t-in␣flow␣" << t_flow << std::endl; // 5
126 std::cout << "cost␣" << cost << std::endl; // 12
```

Marathon revisited (PotW 6.11.17, Exam HS 2017)

Question: How many runners can run along a shortest path without interfering?

Photo: wikipedia.org

Marathon revisited (PotW 6.11.17, Exam HS 2017)

Question: How many runners can run along a shortest path without interfering?

Solution #1: *Without costs*

Solve the edge disjoint path problem in a special graph:

Shortest path graph: keep only edges (u, v) with $\text{dist}(s, t) = \text{dist}(s, u) + \ell(u, v) + \text{dist}(v, t)$

Solution #2: *With costs*

How much does the cheapest flow of size x cost? (cost = length)

What is the largest x such that the min cost of any x-flow $= x \cdot \text{dist}(s, t)$?

Summary: Min Cost Max Flow with BGL

What you should remember from this part:

Minimum Cost Maximum Flow

- is a powerful and versatile modeling tool.
- is a tiebreaker among several maximum flows (but might still not be unique).
- = *maximum* cost maximum flow with negated costs.
- can often be reformulated without negative costs which allows us to use a faster algorithm in BGL (key step in many problems).
- can easily be implemented when starting with our template on the Judge.
- is not harder to use in BGL than the regular network flow algorithms, see BGL Docs.

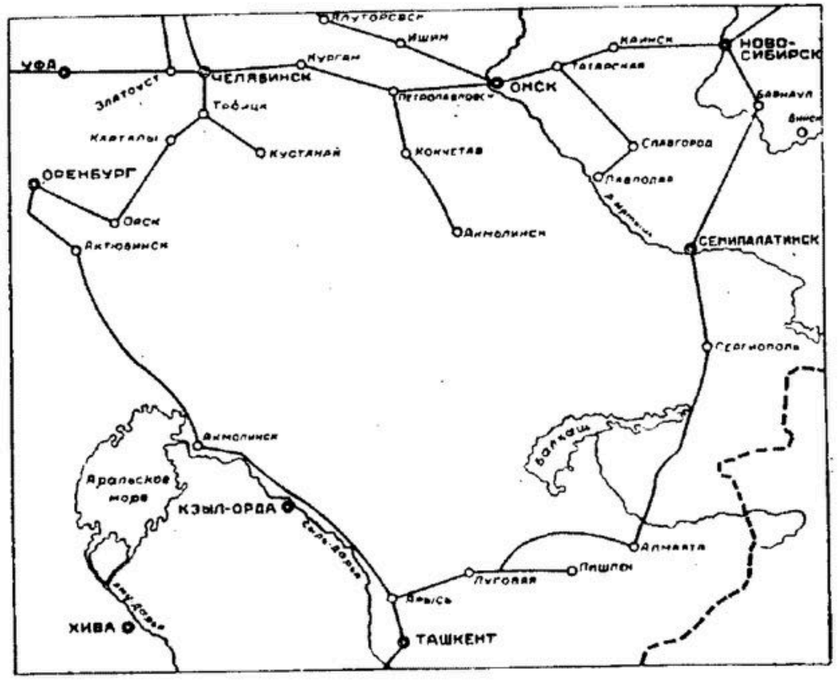
If you are interested in the theory behind these algorithms: (not needed for this course)

- Stanford CS 261 by Prof. Tim Roughgarden, Spring 2016, full lectures on Youtube

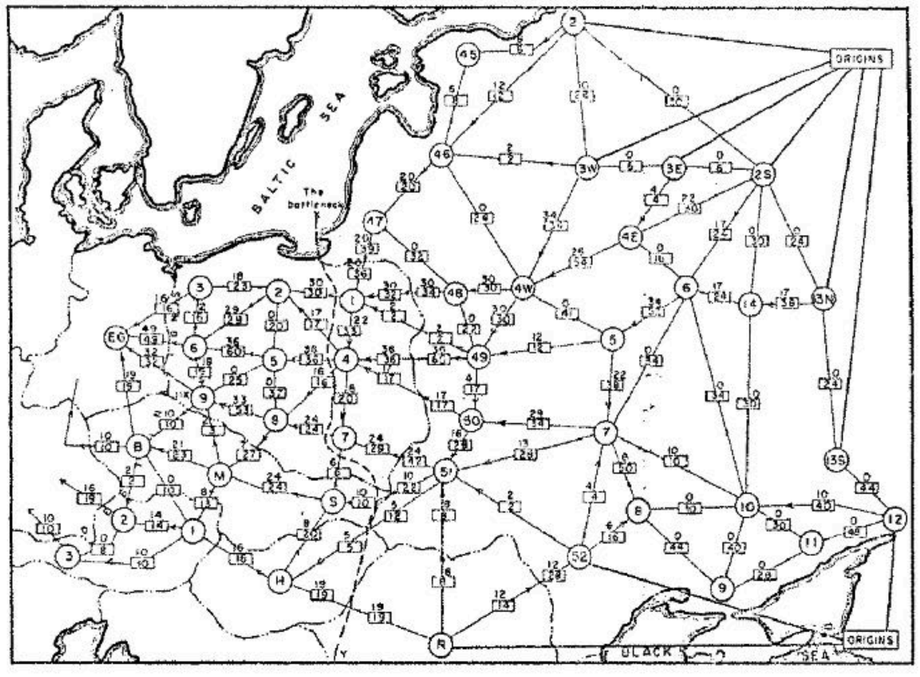
A little bit of history...

What we do in Algolab today was cutting edge research in 1955 ;-)

→ *"On the history of the transportation and maximum flow problems"*
by Alexander Schrijver, Mathematical Programming, 2002



Min Cost Flow, A. N. Tolstoi, 1930
salt & cement transportation
railway network of Soviet Union



Russia/Europe-Min Cut, @ RAND 1955
block max flow of 163'000 tons by air strike
T. E. Harris, F. S. Ross, 1955

First Algorithms

- Cycle canceling idea
  A. N. Tolstoi, 1930
- Greedy flooding
  RAND, A. W. Boldyreff, 1955
- Augmenting paths
  RAND, L. R. Ford Jr., D. R. Fulkerson, 1955

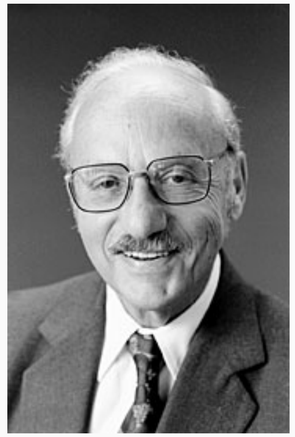
A little bit of history...

What we do in Algolab today was cutting edge research in 1955 ;-)

→ *"On the history of the transportation and maximum flow problems"*
by Alexander Schrijver, Mathematical Programming, 2002

RAND (**R**esearch **AN**d **D**evelopment) ~1940-1960

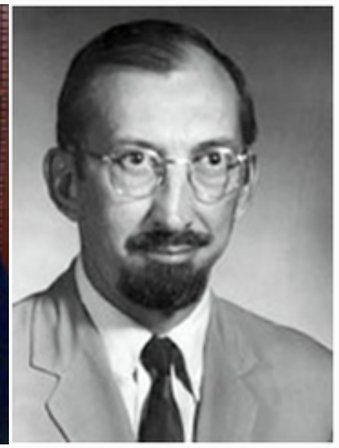
Simplex Method



G. Dantzig
~1947

Network Flow

L. R. Ford Jr.



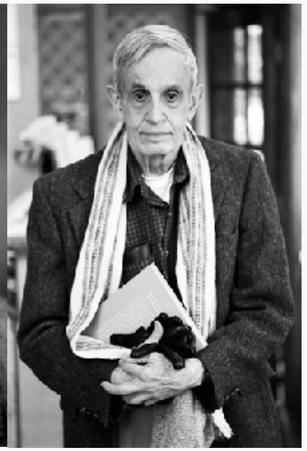
D. R. Fulkerson
~1955

Dynamic Programming

R. Bellman
~1957

and so much more...

J. van Neumann



J. Nash

Photos: stanford.edu, tangrammit.com, wikipedia.org, getty.org, wikipedia.org, wikipedia.org

A little bit of a conclusion...

No new theory and no new tools past this point

But be prepared to combine all your skills:

- On top of a flow problem, do binary search for the answer
- LP formulation vs. flow formulation?
- Some graph problems can be solved greedily (e.g. MST), others not (e.g. flow)
- Dijkstra and MinCostMaxFlows are just “special” dynamic programs
- Find a Min Cost Max Flow formulation where greedy fails for non-unit weights
- Do BFS on Delaunay triangulation or do Union-Find on Euclidean MST
- . . .

Starting next week:

- How to balance reading, solving, coding, debugging under time constraints
- No more problems labeled by topic – figure it out yourself
- In-class exercises on Wednesdays