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

Xun Zou[1]

ETH Zürich

November 17, 2021

[1]based on material from Daniel Graf, Andreas Bärtschi

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

Maximum Flow

- Edge-disjoint paths
- Circulation Problem
- Flow applications

Common Tricks

- Multiple sources/targets
- Undirected edges
- Vertex capacity
- Minimum flow constraint

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

Cuts in directed graphs

- How to disconnect one vertex from another by deleting edges?

Bipartite matchings

- How to compute a maximum matching in a bipartite graph with flow?

Flows with Costs

- What if sending flow comes with a price?

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 in at most one trip.

Result: the number of **edge-disjoint paths** $\Rightarrow$ 4 shops.

- The **bottleneck** between s and t.

Unrealistic condition!

(There are interesting streets in Zürich.)

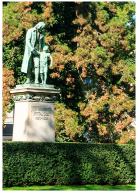
Minimum Cut: Shopping Trip

Start from HB:

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

Condition: **Beautiful roads may be used more than once.**





We may use Bahnhofstrasse up to three times.

Result: the **weighted bottleneck** $\Rightarrow$ 6 shops.

- **minimum cut between s and t.**

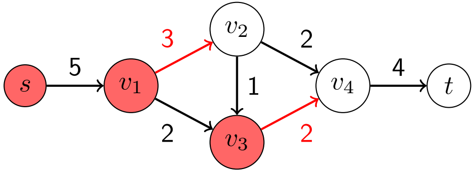
Minimum Cut: Cuts and Flows

$G = (V, E, s, t)$ a flow network.

- $S \subset V$ s.t. $s \in S, t \in V \setminus S (=: T)$, e.g. $S = \{s, v_1, v_3\}$

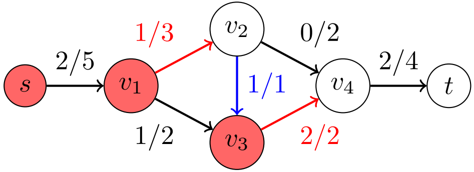
The value of the (S, T)-cut is

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



The value of a flow f from S to T is

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



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

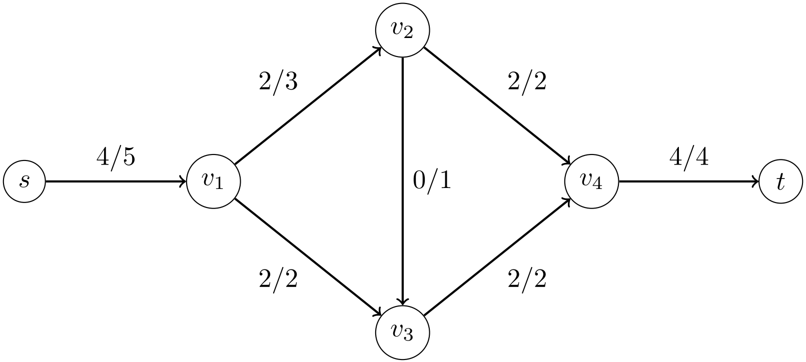
$$|f| = \min_{S\colon\ s\in S, t\notin S} \mathit{cap}(S, V\setminus S).$$

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 cap$(e) - f(e)$, if 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.

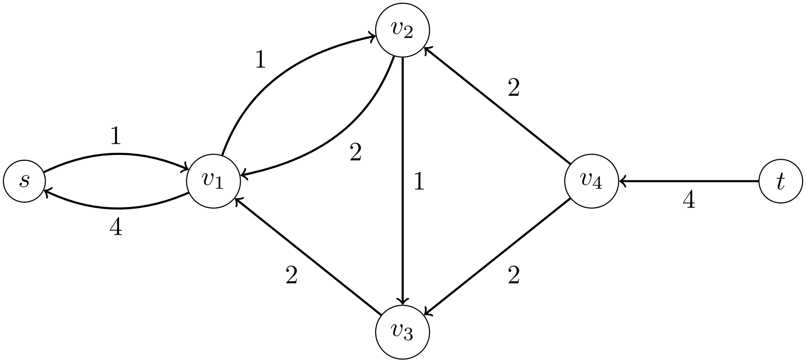
Minimum Cut: Example

Graph G and a maximum flow f.



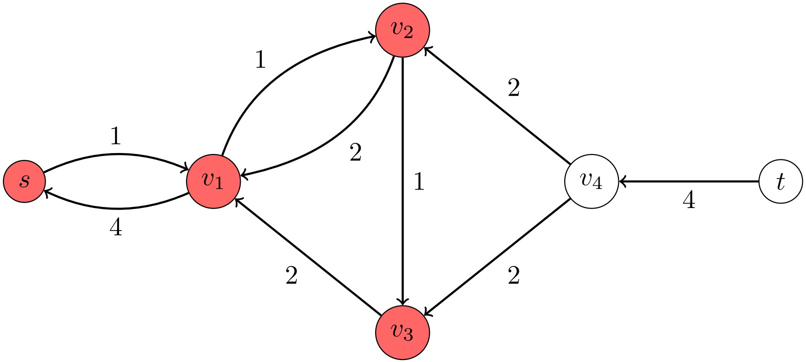
Minimum Cut: Example

Residual graph G_f.



Minimum Cut: Example

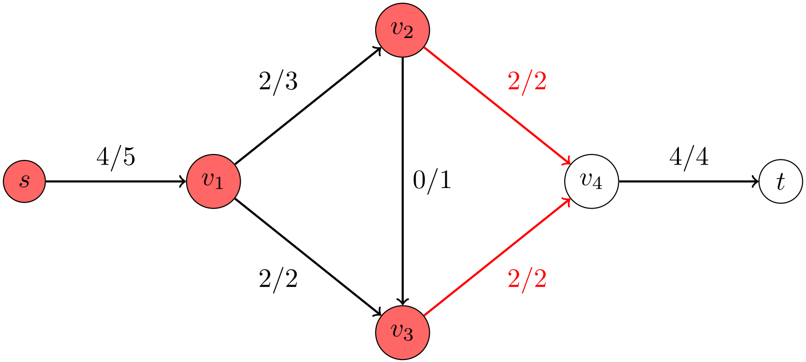
Residual graph G_f.



$S = \{s, v_1, v_2, v_3\}$

Minimum Cut: Example

Residual graph G_f.



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

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. $|f| = \min_{s \in S, t \notin S} \mathit{cap}(S, V \setminus S)$.
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, and the incoming flow to S must be 0.
But then $|f| = f(S, V \setminus S) = \text{cap}(S, V \setminus S)$. □

Minimum Cut: Code

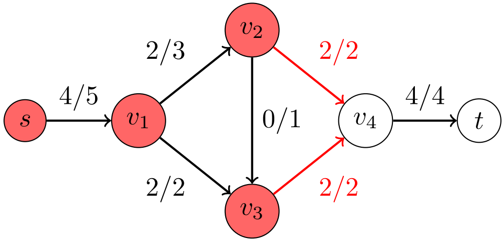
Example code: BFS on the residual graph G_f. See bgl_residual_bfs.cpp on moodle.

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

Minimum cut: Algorithm

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

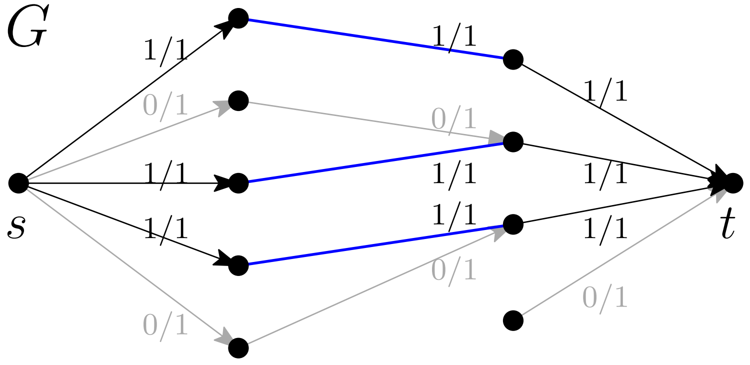
1. Compute maximum flow f and the residual graph G_f.
2. Compute the set S of vertices that are reachable from the source s in G_f.
   - BFS on edges with residual capacity > 0.
3. Output (depending on the task):
   - All vertices in S.
   - All edges going from S to $V \setminus S$.



Note: minimum cuts are not necessarily unique, but the earliest one is.

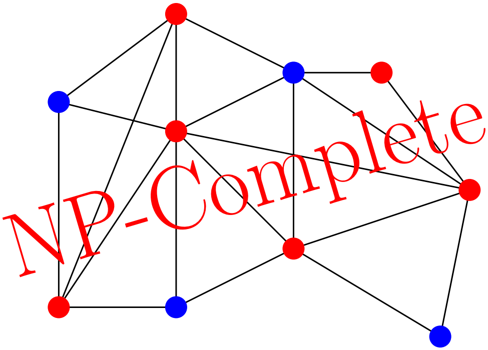
Maximum Matchings: Bipartite Graphs

Maximum Matching: pick as many non-adjacent edges as possible
Flow formulation through vertex capacities / edges-disjoint paths:



Vertex Cover and Independent Set: General Graphs

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



Vertex Cover and Independent Set: Bipartite Graphs

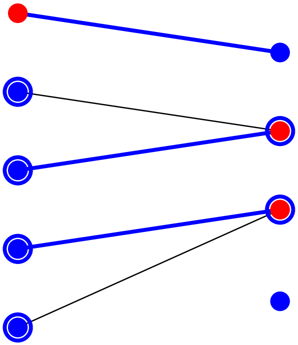
Theorem (König: MinVC and MaxIS are simpler on bipartite graphs!)

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:

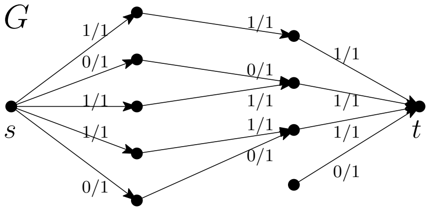
1. Maximum matching M, $V = L \cup R$. Find all unmatched vertices in L, label them as visited.
2. Starting at visited vertices search (BFS) left to right along edges from $E \setminus M$ and right to left along edges from M. Label each found vertex as visited.
3. MinVC – all unvisited in L and all visited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



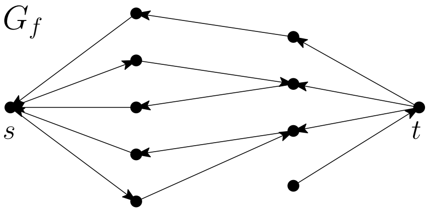
Careful! Step 2 can take several rounds.
Easy Implementation?

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

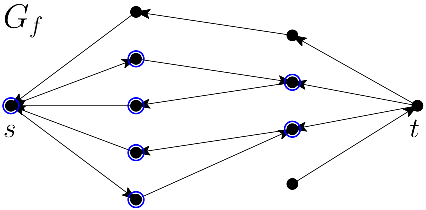
1) Formulate and compute the flow:



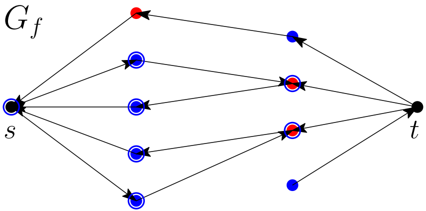
2) Compute the residual graph G_f:



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.

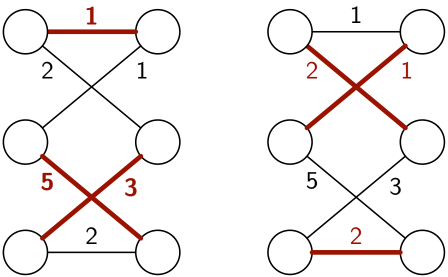
Minimum Cost for a Bipartite Matching

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

Searching for the cheapest among all maximum matchings?
Need for new tools.

two maximum matchings of different cost:



$1 + 5 + 3 = 9$ $\quad$ $2 + 1 + 2 = 5$

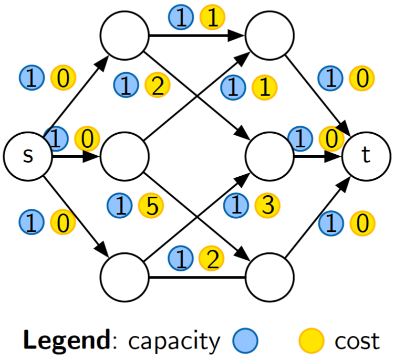
More General Model: Minimum Cost Maximum Flow

Input: a flow network consisting of

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

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

Note: it can model much more than just minimum cost bipartite matching.

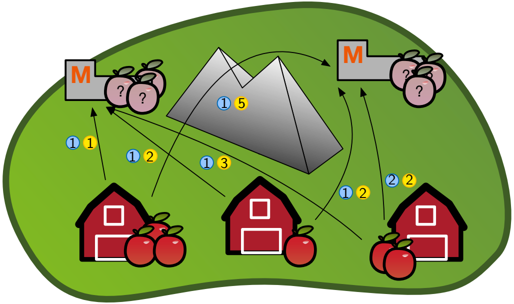


Example: Fruit Delivery

A supermarket wants to schedule fruit deliveries from their farmers to their shops.

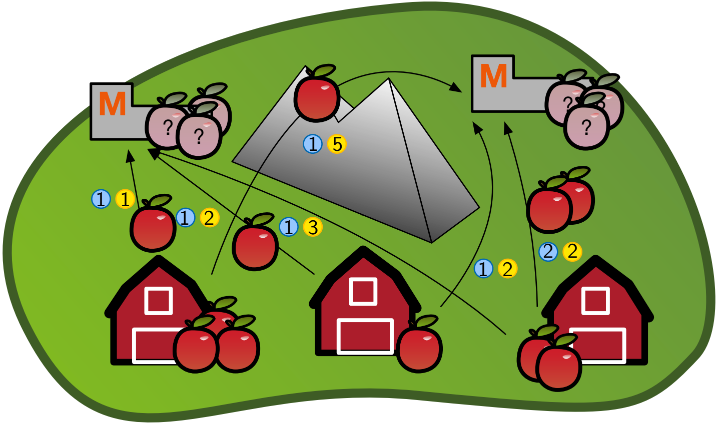
They know all important parameters:

- production per farm [in kg]
- demand per shop [in kg]
- transportation capacity [in kg] and transportation cost [in CHF 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).

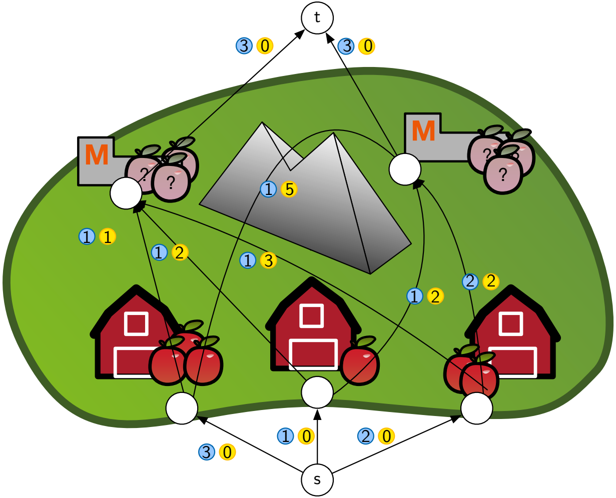
Example: Fruit Delivery



Flow: $2 + 1 + 2 = 5$

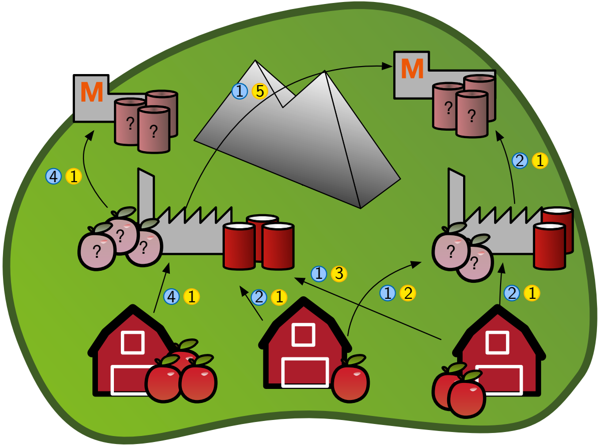
Cost: $1 \cdot 1 + 1 \cdot 5 + 1 \cdot 2 + 2 \cdot 2 = 12$

Example: Fruit Delivery



Extended Example: Canned Fruit Delivery

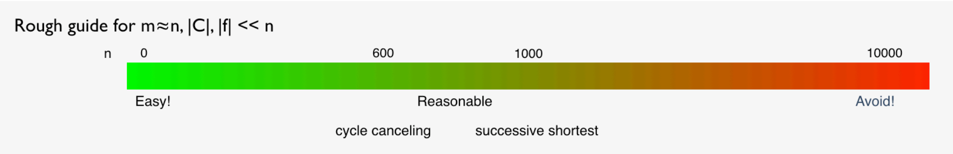
Extension: canned fruit requires transportation to and from a canning factory.



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].



Min Cost Max Flow with BGL

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

```
// Graph Type with nested interior edge properties for Cost 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,
                boost::property <boost::edge_weight_t, long> > > > > graph; // new! weightmap corresponds t

typedef boost::graph_traits<graph>::edge_descriptor            edge_desc;
typedef boost::graph_traits<graph>::out_edge_iterator          out_edge_it; // Iterator
```

Code file: bgl_mincostmaxflow.cpp

Min Cost Max Flow with BGL

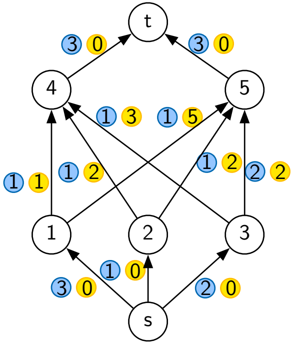
Extending the edge adder:

```cpp
// Custom edge adder class
class edge_adder {
 graph &G;

 public:
  explicit edge_adder(graph &G) : G(G) {}
  void add_edge(int from, int to, long capacity, long cost) {
    auto c_map = boost::get(boost::edge_capacity, G);
    auto r_map = boost::get(boost::edge_reverse, G);
    auto w_map = boost::get(boost::edge_weight, G); // new!
    const edge_desc e = boost::add_edge(from, to, G).first;
    const edge_desc 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;
    w_map[e] = cost;   // new assign cost
    w_map[rev_e] = -cost;   // new negative cost
  }
};
```

Min Cost Max Flow with BGL

Building the graph:

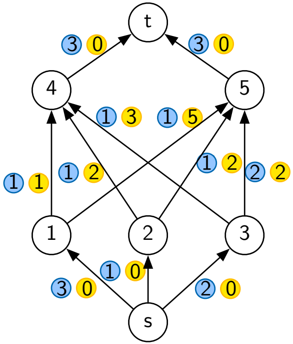
```cpp
const int N=7;
const int v_source = 0;
const int v_farm1 = 1;
const int v_farm2 = 2;
const int v_farm3 = 3;
const int v_shop1 = 4;
const int v_shop2 = 5;
const int v_target = 6;

// Create graph, edge adder class and propery maps
graph G(N);
edge_adder adder(G);
auto c_map = boost::get(boost::edge_capacity, G);
auto r_map = boost::get(boost::edge_reverse, G);
auto rc_map = boost::get(boost::edge_residual_capacity, G);
```



Min Cost Max Flow with BGL

Add the edges:

```
    adder.add_edge(v_source, v_farm1, 3, 0);
    adder.add_edge(v_source, v_farm2, 1, 0);
    adder.add_edge(v_source, v_farm3, 2, 0);

    adder.add_edge(v_farm1, v_shop1, 1, 1);
    adder.add_edge(v_farm1, v_shop2, 1, 5);
    adder.add_edge(v_farm2, v_shop1, 1, 2);
    adder.add_edge(v_farm2, v_shop2, 1, 2);
    adder.add_edge(v_farm3, v_shop1, 1, 3);
    adder.add_edge(v_farm3, v_shop2, 2, 2);

    adder.add_edge(v_shop1, v_target, 3, 0);
    adder.add_edge(v_shop2, v_target, 3, 0);
```



Min Cost Max Flow with BGL

Running the algorithm:

```
// Option 1: Min Cost Max Flow with cycle_canceling
int flow1 = boost::push_relabel_max_flow(G, v_source, v_target);
boost::cycle_canceling(G);
int cost1 = boost::find_flow_cost(G);
std::cout << "-----------------------" << "\n";
std::cout << "Minimum Cost Maximum Flow with cycle_canceling()" << "\n";
std::cout << "flow " << flow1 << "\n"; // 5
std::cout << "cost " << cost1 << "\n"; // 12
```

Min Cost Max Flow with BGL

Running the algorithm:

```
// Option 2: Min Cost Max Flow with successive_shortest_path_nonnegative_weights
boost::successive_shortest_path_nonnegative_weights(G, v_source, v_target);
int cost2 = boost::find_flow_cost(G);
std::cout << "-----------------------" << std::endl;
std::cout << "Minimum Cost Maximum Flow with successive_shortest_path_nonnegative_weights()" << "\n";
std::cout << "cost " << cost2 << "\n"; // 12
// Iterate over all edges leaving the source to sum up the flow values.
int s_flow = 0;
out_edge_it e, eend;
for(boost::tie(e, eend) = boost::out_edges(boost::vertex(v_source,G), G); e != eend; ++e) {
    s_flow += c_map[*e] - rc_map[*e];
}
std::cout << "s-out flow " << s_flow << "\n"; // 5
// Or equivalently, you can do the summation at the sink, but with reversed edge.
int t_flow = 0;
for(boost::tie(e, eend) = boost::out_edges(boost::vertex(v_target,G), G); e != eend; ++e) {
    t_flow += rc_map[*e] - c_map[*e];
}
std::cout << "t-in flow " << t_flow << "\n"; // 5
```

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 Moodle.

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 ;-)
$\rightarrow$ *"On the history of the transportation and maximum flow problems"*
by Alexander Schrijver, Mathematical Programming, 2002

A little bit of conclusion...

No more new theory and 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)
- 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