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

Andreas Bärtschi, Daniel Graf

ETH Zürich

November 11, 2015

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.)

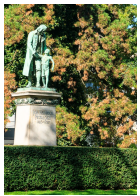
Minimum Cut: Shopping Trip

Start from HB:

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

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





Use Bahnhofstrasse up to three times.

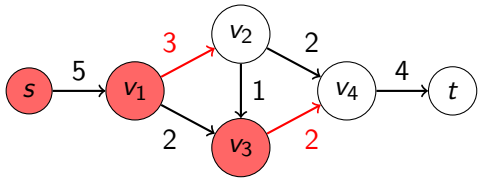
Compute **the weighted bottleneck**, i.e. the **minimum cut between s and t.** $\Rightarrow$ 6 shops.

Minimum Cut: Cuts and Flows

$G = (V, E, s, t)$ a flow network. Let $s \in S, t \in V \setminus S$, for example $\boldsymbol{S = \{s, v_1, v_3\}}$.

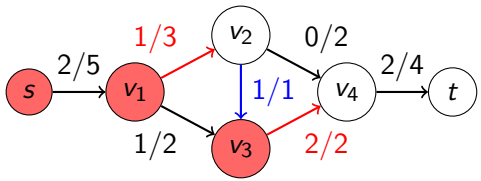
The value of the $(S, V \setminus S)$-cut is

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



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

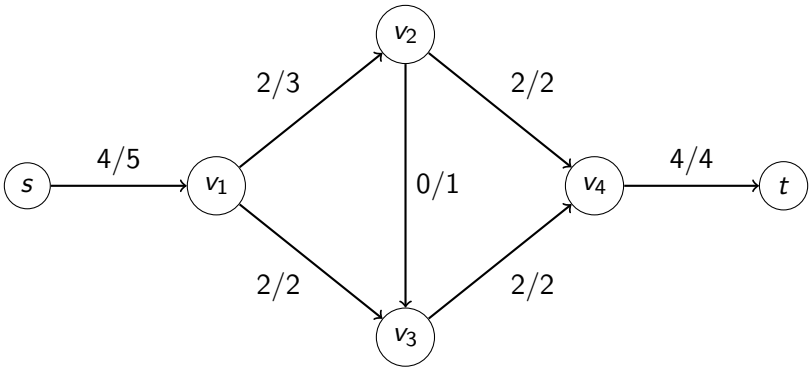
$$|f| = \min_{S:\, s \in S, t \notin S} cap(S, V \setminus S).$$

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.

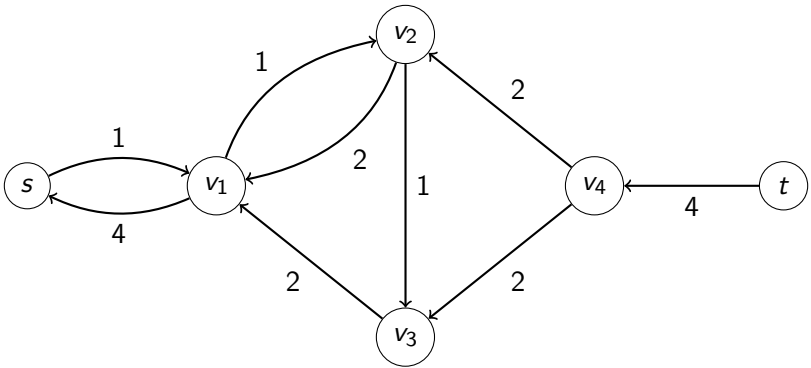
Minimum Cut: Example

Graph G and a maximum flow f.



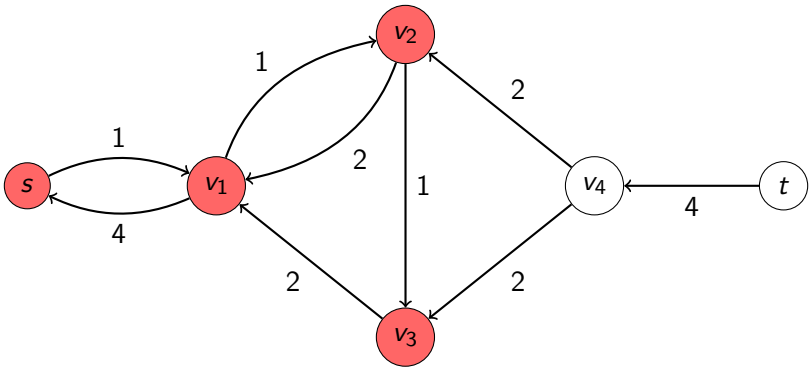
Minimum Cut: Example

Residual graph G_f.



Minimum Cut: Example

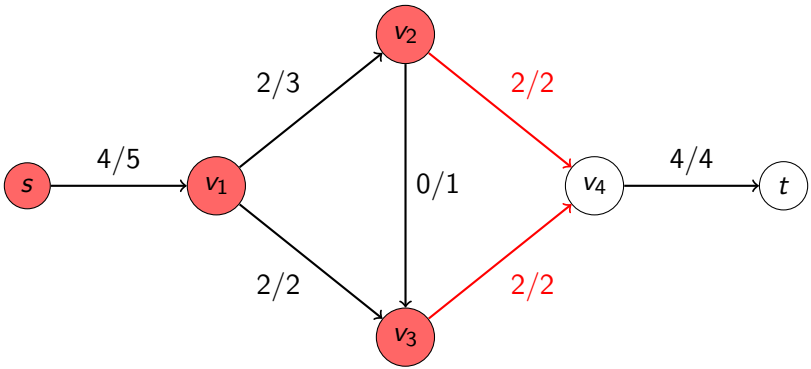
Residual graph G_f.



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

Minimum Cut: Example

Residual graph G_f.



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

Minimum cut: Algorithm

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 $V \setminus S$.

Minimum Cut: Proof of the Maxflow-Mincut-Theorem

Theorem (Maxflow-Mincut-Theorem)

Let f be a 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.

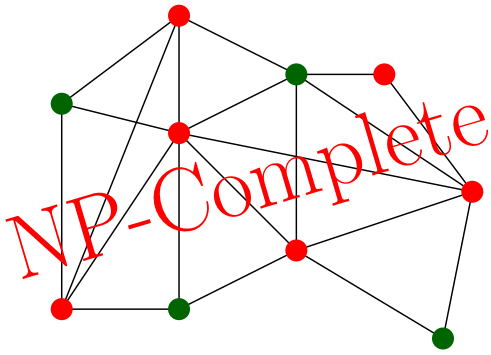
1) $\implies$ 2) f cannot be bigger than $\min_S \text{cap}(S, V \setminus S)$.

2) $\implies$ 3) 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 $|f| = f(S, V \setminus S) = \text{cap}(S, V \setminus S)$. □

Bipartite Matchings: Vertex Cover and Independent Set I

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



Bipartite Matchings: Vertex Cover and Independent Set II

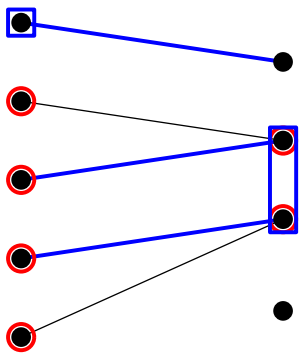
Theorem (König: MinVC is simpler on bipartite graphs!)

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

Algorithm:

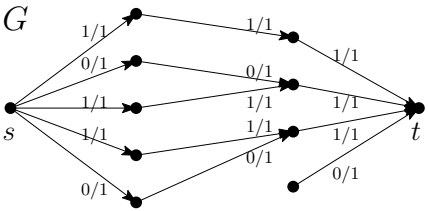
1. Maximum matching M, $V = L \cup R$. Find all unmatched vertices in L, label them as visited.
2. Starting at visited vertices search (BFS) left to right along edges from $E \setminus M$ and right to left along edges from M. Label each found vertex as visited.
3. Minimum vertex cover – all unvisited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

Easy Implementation?

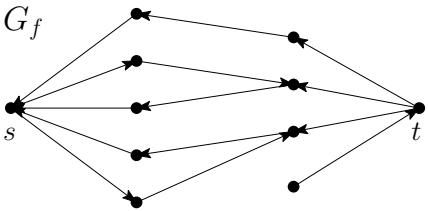


Bipartite Matchings: Modeling via Flow!

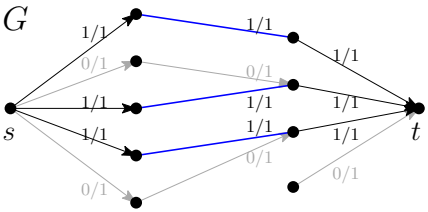
Compute the flow:



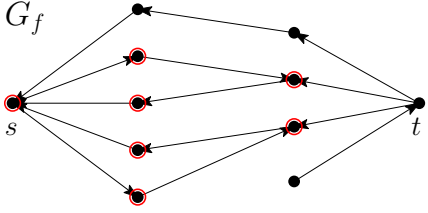
Compute the residual graph G_f:



Find the matching:



Find visited vertices with BFS from s:



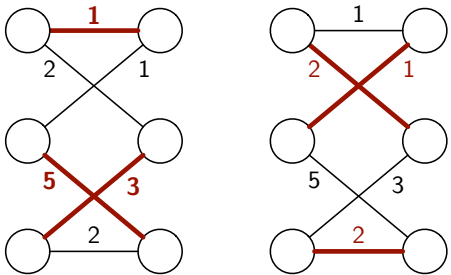
Minimum Cost for a Bipartite Matching

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

- 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 network flow problems anymore.

two maximum matchings of different cost



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

More General: Minimum Cost Maximum Flow

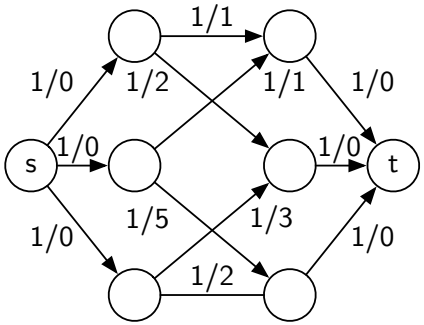
We extend the network flow problem by allowing edge costs that occur per unit of flow.

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.



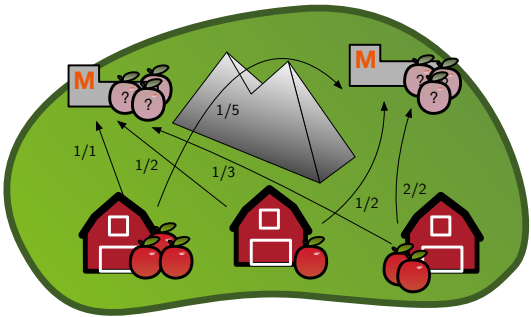
Legend: capacity/cost

Example: Fruit Delivery

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

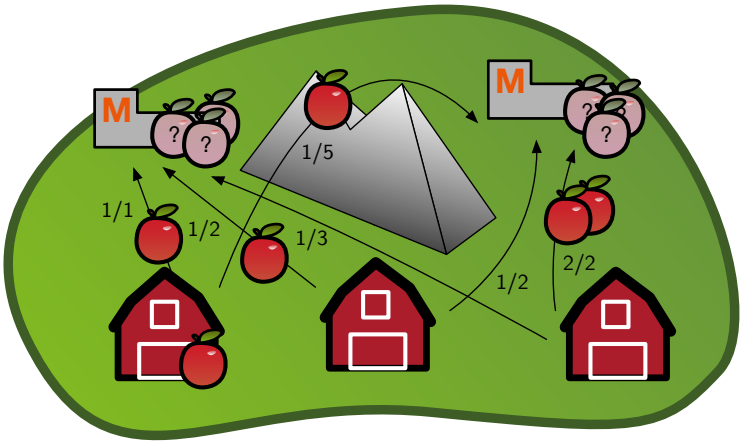
They know all important parameters;

- 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).

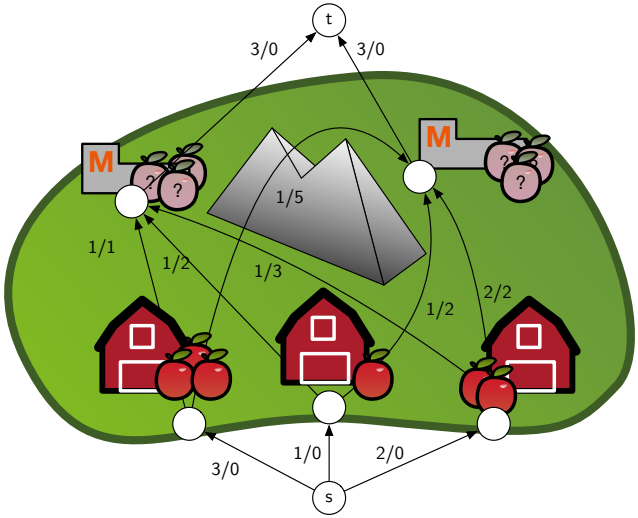
Example: Fruit Delivery



Flow: $2 + 1 + 2 = 5$

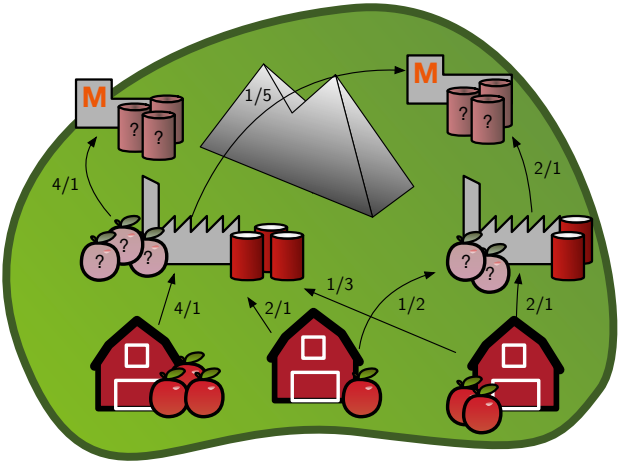
Cost: $1 \cdot 1 + 1 \cdot 5 + 1 \cdot 2 + 2 \cdot 2 = 12$

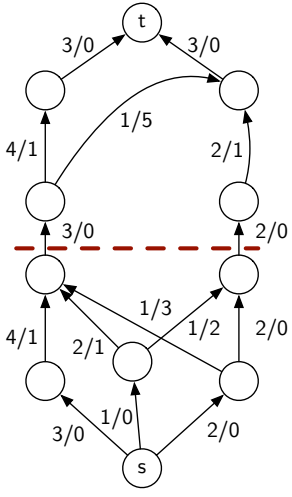
Example: Fruit Delivery



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:

```
1  typedef adjacency_list_traits<vecS, vecS, directedS> Traits;
2  typedef adjacency_list<vecS, vecS, directedS, no_property,
3      property<edge_capacity_t, long,
4          property<edge_residual_capacity_t, long,
5              property<edge_reverse_t, Traits::edge_descriptor,
6                  property <edge_weight_t, long> > > > > Graph; // new!
7
8  typedef property_map<Graph, edge_capacity_t>::type EdgeCapacityMap;
9  typedef property_map<Graph, edge_weight_t >::type EdgeWeightMap; // new!
10 typedef property_map<Graph, edge_residual_capacity_t>::type ResCapacityMap;
11 typedef property_map<Graph, edge_reverse_t>::type ReverseEdgeMap;
12 typedef graph_traits<Graph>::vertex_descriptor Vertex;
13 typedef graph_traits<Graph>::edge_descriptor Edge;
14 typedef graph_traits<Graph>::out_edge_iterator OutEdgeIt;
```

Min Cost Max Flow with BGL

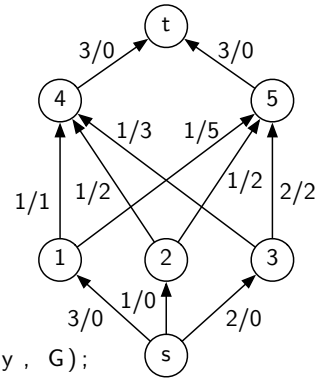
Extending the edge adder:

```
1  struct EdgeAdder {
2      EdgeAdder(Graph & G, EdgeCapacityMap &capacity, EdgeWeightMap &weight,
3       ReverseEdgeMap &rev_edge)
4          : G(G), capacity(capacity), weight(weight), rev_edge(rev_edge) {}
5
6      void addEdge(int u, int v, long c, long w) {
7          Edge e, reverseE;
8          tie(e, tuples::ignore) = add_edge(u, v, G);
9          tie(reverseE, tuples::ignore) = add_edge(v, u, G);
10         capacity[e] = c;
11         weight[e] = w; // new!
12         capacity[reverseE] = 0;
13         weight[reverseE] = -w; // new!
14         rev_edge[e] = reverseE;
15         rev_edge[reverseE] = e;
16     }
17     ...
18 };
```

Min Cost Max Flow with BGL

Building the graph

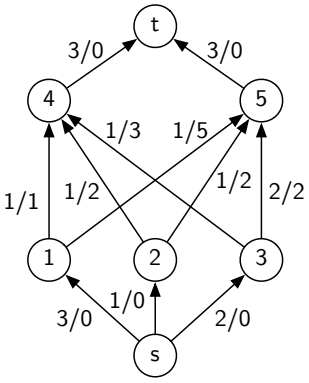
```
1  const int N=7;
2  const int v_source = 0;
3  const int v_farm1 = 1;
4  const int v_farm2 = 2;
5  const int v_farm3 = 3;
6  const int v_shop1 = 4;
7  const int v_shop2 = 5;
8  const int v_target = 6;
9
10 // Create Graph and Maps
11 Graph G(N);
12 EdgeCapacityMap capacity = get(edge_capacity, G);
13 EdgeWeightMap weight = get(edge_weight, G);
14 ReverseEdgeMap rev_edge = get(edge_reverse, G);
15 ResCapacityMap res_capacity = get(edge_residual_capacity, G);
16 EdgeAdder ea(G, capacity, weight, rev_edge);
```



Min Cost Max Flow with BGL

Add the edges:

```
1  ea.addEdge(v_source,v_farm1,3,0);
2  ea.addEdge(v_source,v_farm2,1,0);
3  ea.addEdge(v_source,v_farm3,2,0);
4
5  ea.addEdge(v_farm1,v_shop1,1,1);
6  ea.addEdge(v_farm1,v_shop2,1,5);
7  ea.addEdge(v_farm2,v_shop1,1,2);
8  ea.addEdge(v_farm2,v_shop2,1,2);
9  ea.addEdge(v_farm3,v_shop1,1,3);
10 ea.addEdge(v_farm3,v_shop2,2,2);
11
12 ea.addEdge(v_shop1,v_target,3,0);
13 ea.addEdge(v_shop2,v_target,3,0);
```



Min Cost Max Flow with BGL

Running the algorithm:

```
1 // Option 1:
2 int flow = push_relabel_max_flow(G, v_source, v_target);
3 cycle_canceling(G);
4 int cost = find_flow_cost(G);
5 cout << flow << endl; // 5
6 cout << cost << endl; // 12
```

Min Cost Max Flow with BGL

Running the algorithm:

```
// Option 2: (only for non-negative weights!)
successive_shortest_path_nonnegative_weights(G, v_source, v_target);
int cost = find_flow_cost(G);
int flow = 0;
// Iterate over all edges leaving the source
OutEdgeIt e, eend;
for(tie(e, eend) = out_edges(vertex(v_source,G), G); e != eend; ++e) {
    flow += capacity[*e] - res_capacity[*e];
}
cout << flow << endl; // 5
cout << cost << endl; // 12
```

Min Cost Max Flow with BGL: Summary

What you should remember:

- Minimum cost maximum flow is a powerful and versatile modeling tool.
- With BGL, it is not much harder to use than the regular network flow algorithms.
- Reformulating the problem as a graph without negative costs allows us to use a faster algorithm in BGL.
- Sample implementations are available on the moodle and in the BGL Docs.