We have so far discussed max flows, min cuts, their applications, and several algorithms for it. Today, we consider a generalization of max flow called *minimum-cost* flows, where edges now have costs as well as capacities, and the goal is to find flows with low cost. This problem has a lot of nice theoretical properties, and is also highly practical since it generalizes the max flow problem. We will give two algorithms for the problem that illustrate a nice duality in algorithm design.

Objectives of this lecture

In this lecture, we will:

- Define and motivate the minimum-cost flow problem
- Analyze the properties of minimum-cost flows such as how to determine when they are optimal
- Derive and analyze some algorithms for minimum-cost flows

1 Minimum-Cost Flows

We talked about the problem of assigning groups to time-slots where each group had a list of acceptable versus unacceptable slots. This was the *bipartite matching* problem. A natural generalization is to ask: what about preferences? E.g, maybe group A prefers slot 1 so it costs only \$1 to match to there, their second choice is slot 2 so it costs us \$2 to match the group here, and it can't make slot 3 so it costs \$infinity to match the group to there. And, so on with the other groups. Then we could ask for the *minimum-cost* perfect matching. This is a perfect matching that, out of all perfect matchings, has the least total cost.

The generalization of this problem to flows is called the *minimum-cost flow* problem.

Problem: Minimum-cost flow

We are given a directed graph G where each edge has a *cost*, \$(e), and a capacity, $c(e)$. As before, an *s-t flow* is an assignment of values to edges that satisfy the capacity and conservation constraints, and the *value* of a flow is the net outflow of the source s. The **cost** of a flow is then defined as the sum over all edges, of the cost per unit of flow. That is,

$$cost(f) = \sum_{e \in E} \$(e) f(e).$$

Remark: Cost is per unit of flow

Note **importantly** that the cost is charged *per unit* of flow on each edge. That is, we are not paying a cost to "activate" an edge and then send as much flow through it as we like. This similar problem turns out to be NP-Hard.

The goal of the minimum-cost flow problem is to find feasible flows of minimum possible cost. There are a few different variants of the problem.

- We will consider the **minimum-cost maximum flow** problem, where we seek to find the minimum-cost flow out of all possible maximum flows.
- An alternative formulation is to try to find the minimum-cost flow of value k for some given parameter k, rather than a maximum flow.

These problems can be reduced to each other so they are all equivalent. There are also many other variants of the problem, but we won't consider those for now. Formally, the min-cost max flow problem is defined as follows. Our goal is to find, out of all possible maximum flows, the one with the least total cost. What should we assume about our costs?

- In this problem, we are going to allow *negative costs*. You can think of these as rewards or benefits on edges, so that instead of paying to send flow across an edge, you get paid for it.
- What about negative-cost cycles? It turns out that minimum-cost flows are still perfectly well defined in the presence of negative cycles, so we can allow them, too! Some algorithms for minimum-cost flows however don't work when there are negative cycles, but some do. We will be explicit about which ones do and do not.

Min-cost max flow is more general than plain max flow so it can model more things. For example, it can model the min-cost matching problem described above.

1.1 The residual graph for minimum-cost flows

Let's try to re-use the tools that we used to solve maximum flows for minimum-cost flows. Remember that our key most important tool there was the *residual graph*, which we recall has edges with capacities

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & \text{if } (u,v) \in E, \\ f(v,u) & \text{if } (v,u) \in E. \end{cases}$$

We will re-use the residual graph to attack minimum-cost flows. We need to generalize it, though, because our current definition of the residual graph has no ideas about the costs of the edges. For forward edges e in the residual graph (i.e., those corresponding to $e = (u,v) \in E$), the intuitive value to use for their cost is $\$(e)$, the cost of sending more flow along that edge. What about the reverse edges, though? That's less obvious. Let's think about it, when we send flow along a reverse in the residual graph, we are removing or *redirecting* the flow somewhere else. Since it cost us $\$(e)$ per unit to send flow down that edge initially, when we remove flow on an edge, we can think of getting a *refund* of $\$(e)$ per unit of flow – we get back the money that we originally paid to put flow on that edge. Therefore, the right choice of cost for a reverse edge in the residual graph is $-\$(e)$, the negative of the cost of the corresponding forward edge!

Definition: The residual graph for minimum-cost flows

The residual graph G_f for a minimum-cost flow problem has the same capacities as the original maximum flow problem, but now has costs of $\$(e)$ on a forward edge, and $-\$(e)$ on a back edge. That is, suppose we denote by $\overleftarrow{e}$, the corresponding reverse edge of e in the residual graph. We have

$$c_f(e) = c(e) - f(e), \qquad \$_f(e) = \$(e),$$
$$c_f(\overleftarrow{e}) = f(e) \qquad \$_f(\overleftarrow{e}) = -\$(e)$$

The definitions on the left are the same as regular max flow.

2 An augmenting path algorithm for minimum-cost flows

Now that we have defined a suitable residual graph for minimum-cost flows, we can build an algorithm based on augmenting paths in the same spirit as Ford-Fulkerson. If we just try to pick arbitrary augmenting paths, then it is unlikely that we will find the one of minimum cost, so how should we select our paths in such a way that the costs are accounted for? The most intuitive idea would be to select the *cheapest augmenting path*, i.e., the shortest augmenting path with respect to the costs as the edge lengths. This is not to be confused with the shortest augmenting path algorithm that selects the path with the fewest edges (i.e., the Edmonds-Karp algorithm).

Since the edges are weighted by cost, a breadth-first search won't work anymore. How about our favorite shortest paths algorithm, Dijkstra's? Well, that won't quite work since the graph is going to have negative edge costs (note that even if the input graph does not have any negative costs, the residual graph will, so Dijkstra's will not work here). So, let's use our next-favorite shortest paths algorithm that is capable of handling negative edges: Bellman-Ford! It is important to note here that since the algorithm makes use of shortest path computations, if there is a negative-cost cycle, this algorithm will not work.

Algorithm: Cheapest augmenting paths

Implement Ford-Fulkerson by selecting the *cheapest* augmenting path with respect to residual costs.

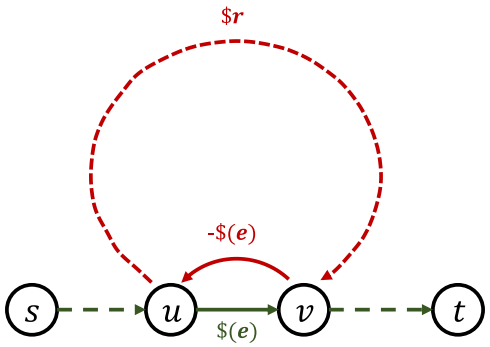
While there exists any augmenting path in the residual graph G_f, find the augmenting path with the cheapest cost and augment as much flow as possible along that path. Since this is just a special case of Ford-Fulkerson, the fact this algorithm finds a maximum flow is immediate, right? Well, not quite. Remember that since shortest paths only exist if there is no negative-cost cycle, we need to prove that our algorithm never creates one after performing an augmentation, or the next iteration's shortest path computation will fail.

Theorem 1: Maintenance of minimum-cost flows

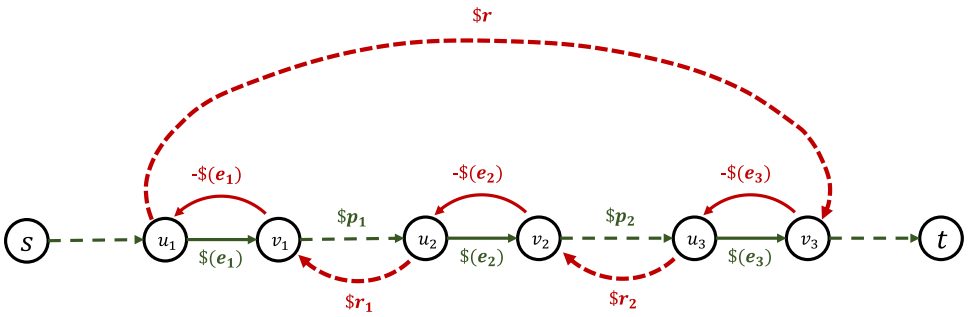
If the original graph has no negative cycles, at each iteration of the cheapest augmenting paths algorithm, the residual graph contains no negative-cost cycles.

Proof. We can proceed by induction on the iterations of the algorithm. For the cheapest augmenting paths algorithm, we don't allow negative-cost cycles in G, therefore the initial residual graph, which is just G, contains no negative-cost cycles, which establishes a base case. We now need to show that if at some step, the current residual graph contains no negative-cost cycles, then augmenting along a cheapest path produces a new residual graph that still contains no negative-cost cycles.

Consider the residual graph that contains no negative-cost cycles at the beginning of an iteration of the algorithm. We need to show that after augmenting along a cheapest augmenting path that there are still no negative-cost cycles in the residual graph. Suppose for the sake of contradiction that an augmentation introduces a negative cycle. Since the augmenting path creates reverse edges in the residual graph, at least one of these reverse edges must intersect the newly created negative-cost cycle.



Suppose the corresponding edge e had cost $\$(e)$, and the remainder of the cycle has cost r. Since the cycle has a negative weight, we have $-\$(e) + r < 0$, but this implies that $r < \$(e)$, and hence it would have been a *cheaper path* to travel around the cycle instead of using the edge e! This is a contradiction, so we're done right? Unfortunately not. This only covers the simplest case, where the cycle intersects the path only once, but it might actually intersect the path multiple times.



In general, when there are k intersecting edges as in the diagram above, the cost of the cycle is

$$r + \sum_{i=1}^{k-1} r_k - \sum_{i=1}^{k} \$(e_i) < 0$$

Now observe that the cycles formed by the cost $\$r_i$+$\$p_i$ edges were already in the graph before the augmenting path was added (they do not consist of any newly introduced edges), so they can not be negative-cost cycles. Therefore $p_i + r_i \geq 0$ for all i, or equivalently $r_i \geq -p_i$. So, the cost of the negative cycle is

$$r - \sum_{i=1}^{k-1} p_i - \sum_{i=1}^{k} \$(e_i) \leq r + \sum_{i=1}^{k-1} r_k - \sum_{i=1}^{k} \$(e_i) < 0$$

Rearranging, we get that

$$r < \sum_{i=1}^{k-1} p_i + \sum_{i=1}^{k} \$(e_i),$$

but the sum on the right is the cost of the path from u_1 to v_k, so this implies that replacing this path with r gives us a cheaper augmenting path that we could have used in the first place. This is a contradiction, and hence we can conclude that the resulting residual graph has no negative cost cycles. □

So, we have successfully proven that the algorithm will terminate, and since it is a special case of Ford-Fulkerson, it terminates with a maximum flow. We can also argue about the runtime using the same logic that we used for Ford-Fulkerson. At every iteration of the algorithm, at least one unit of flow is augmented, and every iteration has to run the Bellman-Ford algorithm which takes $O(nm)$ time. Therefore, the worst-case running time of cheapest augmenting paths is $O(nmF)$, where F is the value of the maximum flow, assuming that the capacities are integers.

What we still have to prove, however, is that this algorithm truly finds a *minimum-cost* flow.

3 An optimality criteria for minimum-cost flows

When studying maximum flows, our ingredients for proving that a flow was maximum were augmenting paths and the minimum cut. We'd like to find a similar tool that can be used to prove that a minimum-cost flow is optimal. First, let's be specific about what we mean by optimality.

Definition: Cost optimality of flows

We will say that a flow f is *cost optimal* if it is the cheapest of all possible flows *of the same value.*

That is to say we don't compare the costs of different flows if they have different values. Our goal is to find a tool to help us analyze the cost optimality of a flow. To do so, we're going to think about what kinds of operations we can do to modify a flow and change its cost *without* changing its value.

When finding maximum flows, our key tool was the *augmenting path*. If there existed an augmenting path in G_f, then by adding flow to it, we could transform a given flow into a more optimal one that was still feasible because adding flow to an s-t path preserved the flow conservation condition, and didn't violate the capacity constraint as long as we added an appropriate amount of flow. Therefore, the existence of an augmenting path proved that a flow was not maximum, and we were able to later prove that the lack of existence of an augmenting path was sufficient to conclude that a flow was maximum. We want to discover something similar for cost optimality of a flow. Augmenting paths were just one way to modify a flow while keeping it feasible. I claim that there is one other way that we can modify a flow while still preserving feasibility, but that also doesn't change its value. Instead of adding flow to an s-t path, what if we instead add flow to a *cycle* in the graph? Let's call this an *augmenting cycle*.

Since a cycle has an edge in and an edge out of every vertex, adding flow to a cycle in the residual graph preserves flow conservation, and hence feasibility. Furthermore, since the same amount of flow goes in and out of each vertex, the flow value is unaffected! However, adding flow to a cycle may in fact change the cost of the flow, which is what we wanted! Suppose the costs of the edges $e_1, e_2, ..., e_k$ on the cycle are $\$_1, \$_2, ..., \$_k$ for some cycle of length k. Then, adding Δ units of flow to the cycle will change the cost by

$$\sum_{i=1}^{k} \Delta \cdot \$_i = \Delta \cdot \sum_{i=1}^{k} \$_i.$$

Now, note that $\sum \$_i$ is just the weight of the cycle! So, we can say that if we add Δ units of flow to a cycle of weight W, then the cost of the flow changes by $\Delta \cdot W$. If $\Delta \cdot W$ is negative, then we have just shown that the cost of the flow *can be decreased*, so it wasn't optimal!

It turns out that this is the key ingredient for analyzing minimum-cost flows. Even better, we are getting a two-for-one deal because a lack of negative-cost cycles was what we already argued was required for our cheapest augmenting path algorithm to produce a maximum flow. It turns out that this same condition allows us to prove that the flow is cost optimal.

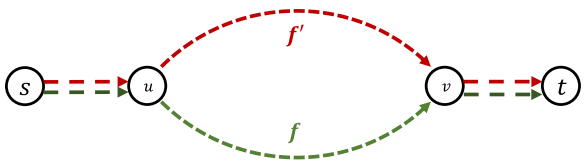
Theorem 2: Cost optimality and negative cycles

A flow f is cost optimal if and only if there is no negative-cost cycle in G_f.

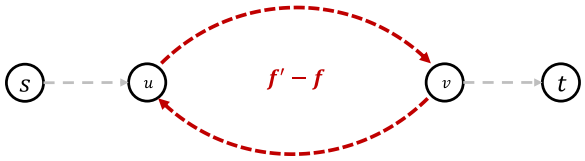
Proof. We pretty much already proved the first direction in the previous paragraph. Suppose that there exists a negative cost cycle in the residual graph. Then by adding flow to this cycle, the value of the flow doesn't change, but the cost changes by $\Delta \cdot W$, where Δ is the amount of flow we can add, and W is the cost/weight of the cycle. Since W is negative, the cost decreases, and hence the flow f was not cost optimal.

Now suppose that a flow f is not cost optimal. We need to prove that there exists a negative-cost cycle in its residual graph G_f. This case is much trickier. Since f is not cost optimal, there exists some other flow f' of the same value but which has a cheaper cost. Let's now consider the flow $f' - f$. Hang on, what does this even mean? Let's work that out. For each edge (u, v), consider two cases:

- if $f'(u, v) - f(u, v) \geq 0$, then we will say that the edge (u, v) has $f'(u, v) - f(u, v)$ flow,
- if $f'(u, v) - f(u, v) < 0$, then we will say that the *reverse* edge (v, u) has a flow of $f(u, v) - f'(u, v)$.



Lets try to make this clearer with an example. Here is a network with two flows: red, above; and green, below, such that the two overlap on the path from s to u and v to t, but take a different path from u to v. Suppose they both have value one. If we take the difference of these flows, then the flows from s to u and v to t cancel, but the flow from u to v gets reversed, so we end up with the following.



We've created a flow that just contains a cycle! Now the claim is that this is what always happens: the difference between two flows of the same value is a collection of (possibly multiple) cycles. But how would we prove this, and how do we know that it is even a valid flow? Well, since both f' and f are feasible flows, they satisfy the flow conservation property, and hence their difference $f' - f$ also satisfies the flow conservation property (for example, if some vertex has 5 flow in and out in f' and 3 flow in and out in f, then their difference has 2 flow in and out). What is the value of this flow? By assumption, the values of f' and f are the same, hence their net flows out of s are the same. By taking the difference between the two, we can see that the net flow out of s in $f' - f$ is actually **zero**. So, we have a flow of value zero, but it isn't the all-zero flow, so what does this look like? Well, every vertex *including the source and sink* have flow in equal to flow out, which means that the flow is indeed a *collection of cycles*. For this reason, such a flow is often called a *circulation*.

What about capacity constraints? Actually, it looks like this flow might not satisfy the capacity constraints of G, because it could send flow along a reverse edge that doesn't even exist in G. What actually matters to us is that $f' - f$ is a valid flow **in the residual graph** G_f. If $f'(e) - f(e) \geq 0$, then the forward edge e has flow $f'(e) - f(e) \leq c(e) - f(e)$ which is the definition of the residual capacity $c_f(e)$. Alternatively, if $f'(e) - f(e) < 0$, then the reverse edge $\vec{e}$ has flow $f(u,v) - f'(u,v) \leq f(u,v)$, which is the definition of the residual capacity of $c_f(\vec{e})$. Therefore, $f' - f$ is a feasible flow in the residual graph!

Lastly, we should think about the *cost* of this flow. Some algebra will show us that

$$\text{cost}(f' - f) = \text{cost}(f') - \text{cost}(f).$$

Since we assumed that $\text{cost}(f') < \text{cost}(f)$, this must be a negative cost. So, we have a collection of cycles of flow whose total cost is negative, therefore at least one of those cycles must have a negative cost. Since $f' - f$ is feasible in the residual graph, this negative-cost cycle exists in the residual graph. □

Corollary: Optimality of cheapest augmenting paths

Since we already argued in Theorem 1 that the cheapest augmenting path algorithm never creates a negative-cost cycle in the residual graph, this theorem proves that the resulting flow is also cost optimal. Therefore we can conclude that the cheapest augmenting path algorithm successfully finds the minimum-cost maximum flow!

4 Cycle canceling: Another algorithm for min-cost flow

Finally, lets talk about another algorithm for minimum-cost flows that demonstrates a really nice duality in algorithm design. We're going to derive and analyze an algorithm that solves the min-cost flow problem in rather the "opposite" way to the cheapest augmenting paths problem.

At a high level, the cheapest augmenting paths algorithm works by beginning with a flow that is cost optimal (the all zero flow, assuming no negative-cost cycles) but not maximum, and then iteratively making the flow more maximum via augmenting paths while maintaining the invariant that it is always cost optimal. What if we did the opposite and started with a flow that is maximum but not cost optimal, and then iteratively made it cheaper while maintaining the invariant that it is maximum? That is the idea of *cycle canceling*.

Recall that a flow is maximum if and only if there are no augmenting paths (we proved this implicitly while analyzing the Ford-Fulkerson algorithm). Theorem 2 gives an analogous tool for cost optimality. A flow is cost optimal if and only if it has no negative cycles. So, just like the Ford-Fulkerson algorithm works by identifying augmenting paths and adding flow to them until none exist, we can write an algorithm that does the same but for negative-cost cycles. We just need an algorithm that finds a negative-cost cycle or reports that none exist. Luckily, the Bellman-Ford algorithm does exactly this. If the algorithm finds a negative cycle, it can use it as an *augmenting cycle* to to decrease the cost of the flow. Remember that adding flow to a cycle maintains feasibility, doesn't affect the flow value, and if the cost of the cycle is negative, it results in a decrease to the cost of the flow.

Algorithm: Cycle-canceling for minimum cost flows

```
find a maximum flow f with any max flow algorithm
while there exists a negative-cost cycle in G_f
  augment the maximum possible amount of flow along the cycle
```

A great thing about this is that we don't really need to do any more analysis either. Theorem 2 immediately proves that this algorithm, if it terminates, is correct. Another huge plus of this algorithm is that it works even when there are negative-cost cycles in the original graph!

Remark: Minimum-cost flow with negative-cost cycles

Unlike the cheapest augmenting path algorithm, the cycle-canceling algorithm works when the input graph has negative-cost cycles!

How can we prove that it terminates? Well, lets proceed similarly to Ford-Fulkerson where we simplify a bit and assume that the costs are all integers. If so, every augmenting cycle lowers the cost by at least one, so as long as the problem is well defined (the minimum cost is finite), the algorithm will eventually terminate!

Theorem: Running time of cycle canceling

Assume that the graph has integer capacities between 1 and U, and integer costs between $-C$ and C. Then the cycle-canceling algorithm runs in $O(nm^2UC)$.

Proof. Then, the maximum possible cost of any flow is

$$\sum_{e \in E} c(e) \max(0, \$(e)) \leq UCm,$$

Since the minimum cost could be negative, the cycle-canceling algorithm could take up to $2UCm$ iterations. Each iteration takes $O(nm)$ time with Bellman-Ford, so the total runtime is at most $O(nm^2UC)$. □

Just like we improved the Ford-Fulkerson algorithm from non-polynomial time to polynomial time by picking better augmenting paths rather than arbitrary ones, the cycle canceling algorithm can also be improved to polynomial time by picking better cycles rather than arbitrary ones, but we don't have time to cover the details here. There are also other techniques for solving minimum-cost flow problems in polynomial time and for non-integer capacities and costs, so just know that it can be done even though we won't cover how.