Reduction

Reduction is an extremely important tool to "measure" the difficulty of a problem. We will not go into technical details, such as Turing machine, NP, polynomial time certificate and so on. We just show by some examples how to prove that a problem is NP-hard.

Independent Set (reduction from Clique)

Given a graph, Independent Set asks the maximum number of vertices so that *no* two of them are connected. Clique asks the maximum number of vertices so that *every* two of them are connected.

If I know already that Clique is an NP-hard problem, how could I show that Independent Set is *at least as hard*? So we can do a reduction as follows. Given an instance $G = (V, E)$ of Clique problem, we create another instance $\overline{G} = (V, \overline{E})$, where the latter graph is simply the complement of the former. It should be clear: there is a clique of size k in the first graph if and only if there is an independent set in the second graph.

Independent Set (reduction from 3SAT)

This time we want to show that Independent Set is at least as hard as 3SAT. Remember that in the latter problem, we are given a boolean formula ϕ, which takes the form of conjuctive form of clauses of 3 literals: for instance $(v_1 \vee \overline{v}_2 \vee v_5) \wedge ((\overline{v}_1 1 \vee \overline{v_4} \vee v_7) \wedge \cdots$ ...

The reduction is as follows. For each clause, we create a triangle, where each vertex in a triangle representing a literal. We will also connect the triangles in the following way. If v_{i1} represents a literal contained in clause i and v_{j2} represents a literal contained in another difficult clause j, and the two literals are the opposite of each other, namely, v and $\overline{v}$, then we connect v_{i1} and v_{j2}.

After we have constructed the graph G in this manner, we claim that the 3SAT formula ϕ has a "YES" solution if only if G has an independent set of size m (where m is the number of clauses in ϕ). This follows from the fact that we can take at most one vertex in each triangle (clause) and we never choose two literals that are the opposite of each other.

Max Flow: Ford-Fulkerson

(You have learned about max flow in your L3 course. Here is a quick summary to refresh your memory)

Let $G = (V, E)$ be a directed graph where every edge e has an integer capacity $c_e > 0$. Two special nodes $s, t \in V$ are called source and sink, all other nodes are called internal. We suppose that no edge enters s or leaves t. A flow is a function f on the edges such that: $0 \leq f(e) \leq c_e$ holds for all edges e (capacity constraints), and $f^+(v) = f^-(v)$ holds for all internal nodes v (conservation constraints), where we define

$f^-(v) := \sum_{e=(u,v)\in E} f(e)$ and $f^+(v) := \sum_{e=(v,u)\in E} f(e)$. (As a menominic aid: $f^-(v)$ is consumed by node v, and $f^+(v)$ is generated by node v.) The value of the flow f is defined as $val(f) := f^+(s)$. The Maximum Flow problem is to compute a flow with maximum value.

For any flow f in G (not necessarily maximum), we define the residual graph G_f as follows. G_f has the same nodes as G. For every edge e of G with $f(e) < c_e$, G_f has the same edge with capacity $r_e = c_e - f(e)$, called a forward edge. The difference is obviously the remaining capacity available on e. For every edge e of G with $f(e) > 0$, G_f also has the opposite edge with capacity $r_e = f(e)$, called a backward edge. By virtue of backward edges we can "undo" any amount of flow up to $f(e)$ on e by sending it back in the opposite direction.

Now let P be any simple directed $s-t$ path in G_f, and let b be the smallest residual capacity of all edges in P. For every forward edge e in P, we may increase $f(e)$ in G by b, and for every backward edge e in P, we may decrease $f(e)$ in G by b. It is not hard to check that the resulting function f' on the edges is still a flow in G. We call f' an augmented flow, obtained by these changes. Note that $val(f') = val(f) + b > val(f)$.

Now the basic Ford-Fulkerson algorithm works as follows: Initially let $f := 0$. As long as a directed $s-t$ path in G_f exists, augment the flow f (as described above).

To prove that Ford-Fulkerson outputs a maximum flow, we must show: If no $s-t$ path in G_f exists, then f is a maximum flow.

The proof is done via another concept of independent interest: An $s-t$ cut in $G = (V, E)$ is a partition of V into sets A, B with $s \in A, t \in B$. The capacity of a cut is defined as $c(A, B) := \sum_{e=(u,v):u\in A,v\in B} c_e$.

For subsets $S \subset V$ we define $f^+(S) := \sum_{e=(u,v):u\in S,v\notin S} f(e)$ and $f^-(S) := \sum_{e=(u,v):u\notin S,v\in S} f(e)$. Remember that $val(f) = f^+(s) - f^-(s)$ by definition. (Actually we have $f^-(s) = 0$ if no edge goes into s.) We can generalize this equation to any cut: $val(f) = \sum_{u\in A}(f^+(u) - f^-(u))$, which follows easily from the conservation constraints. When we rewrite the last expression for $val(f)$ as a sum of flows on edges, then, for edges e with both nodes in A, terms $+f(e)$ and $-f(e)$ cancel out in the sum. It remains $val(f) = f^+(A) - f^-(A)$. It follows $val(f) \leq f^+(A) = \sum_{e=(u,v):u\in A,v\notin A} f(e) \leq \sum_{e=(u,v):u\in A,v\notin A} c_e = c(A, B)$. In words: The flow value $val(f)$ is bounded by the capacity of any cut (which is also intuitive).

Next we show that, for the flow f returned by Ford-Fulkerson, there exists a cut with $val(f) = c(A, B)$. This implies that the algorithm in fact computes a maximum flow.

Clearly, when the Ford-Fulkerson algorithm stops, no directed $s-t$ path exists in G_f. Now we specify a cut as desired: Let A be the set of nodes v such that some directed $s-v$ path is in G_f, and $B = V\setminus A$. Since $s \in A$ and $t \in B$, this is actually a cut. For every edge (u, v) with $u \in A$, $v \in B$ we have $f(e) = c_e$ (or v should be in A). For every edge (u, v) with $u \in B$, $v \in A$ we have $f(e) = 0$ (or u should be in A because of the backward edge (v, u) in G_f). Altogether we obtain $val(f) = f^+(A) - f^-(A) = f^+(A) = c(A, B)$. In words: The flow value $val(f)$ equals the capacity of a minimum cut (which is still intuitive).

The last statement is the famous Max-Flow Min-Cut Theorem. It should be noted that in case all capacities $c_e \in Z_{>0}$, then there is a *integral* max flow, i.e., all $f(e)$ are integers. (Why?) This is a useful property for some applications.

We remark that by original Ford-Fulkerson algorithm may *not* stop. But in case that all edge capacities are integers, it terminates in $O(m^2C)$ time, where $C = \max_{e \in E} c_e$. One needs $O(m)$ time to build the residula graph and find a $s-t$ path in it. How many augmentions do we need? As each time we augment at least 1 unit and the flow value cannot be larger than $O(mC)$ (why?) We have the claimed complexity.

Max Flow: Edmond-Karp

The Edmonds-Karp algorithm is motivated by that "pathological" example that we have seen in class. Its idea is very simple: augment along the shortest path in the residual network G_f.

In the following, we write $P_1, P_2, \cdots$ as the sequence of paths that we have found in the residual network. Let f_i denotes the current flow after we have augmented (in sequence) $P_1, \cdots, P_{i-1}$. Also let $E(P_i)$ denote the set of edges used by P_i. The following lemma is crucial: it implies that the paths P_i grow in length monotonically.

Lemma 1. *In Edmonds-Karp algorithm, let $P_1, \cdots$ be the sequence of augmenting paths. Then*

1. $|E(P_k)| \leq |E(P_{k+1})|$.
2. *If P_k and P_l share a pair of reverse edges and $k < l$, then* $|E(P_k)| + 2 \leq |E(P_l)|$.

Proof. For (1), we define a graph H as the union of P_k and P_{k+1}, after we have removed the pairs of reverse edges of these two paths. Here we observe that every s-t path in H must also be an augmenting path in G_{f_k}: if an edge is in P_k, it certainly is in G_{f_k}. But how about an edge in P_{k+1}? Couldn't it be a new edge absent in G_{f_k}? But this cannot happen. If it is a new edge, it must have popped up as the reverse edge of some edge along P_k. But such a pair of reverse edges are removed from H, by our construction.

Let us add two directed edges from t to s in H. Observe that H is now Eulerian (every vertex has the same outgoing and incoming degrees). So there are two disjoint circuits containing the two added directed edges from t to s, implying that we have two directed s-t paths Q_1 and Q_2 in H. Remember that Edmonds-Karp algorithm chooses the shortest path. So these two paths cannot be shorter than P_k. As a consequence, in G_{f_k},

$$2|E(P_k)| \leq |E(Q_1)| + |E(Q_2)| \leq |E(H)| \leq |E(P_k)| + |E(P_{k+1})|,$$

hence the proof of (1).

For (2), we will prove the following claim (which easily implies the proof thanks to (1)).

Claim 1. *Suppose that before P_l, P_k is the latest path that uses a reverse edge of P_l (in other words, none of $P_{k+1}, \cdots P_{l-1}$ uses a reverse edge of P_l). Then* $|E(P_k)| + 2 \leq E(P_l)$.

The proof of the claim is very similar to the above proof of (1). Let H be the union of P_k and P_l after we have removed their pairs of reverse edges. Again we make the observation that every s-t path in H is also a path in $G(f_k)$ (this is a bit more subtle than the last time: try to convince yourself). Then again we have two paths Q_1 and Q_2 in H and we derive

$$2|E(P_k)| \leq |E(Q_1)| + |E(Q_2)| \leq |E(H)| \leq |E(P_k)| + |E(P_{k+1})| - 2,$$

where the extra -2 term comes from the fact P_k and P_l share at least a pair of reverse edges.

□

We can now prove the main theorem.

Theorem 1. *Edmonds-Karp algorithm augments $O(nm)$ times, implying a total running time of $O(nm^2)$.*

Proof. Remember every augmenting path is associated with a bottleneck edge. How many times an edge in the residual network can be a bottleneck? Suppose that e is the bottleneck of augmenting paths $P_{i_1}, P_{i_2}, \cdots$. Here we observe that after path P_{i_k} is augmented, the edge e disappears. Then how could it re-appear to be used by $P_{i_{k+1}}$? There must exist another path P_j, where $i_k < j < i_{k+1}$ such that P_j augments along the reverse edge of e. By Lemma 1(2), we then know that

$$|E(P_{i_k})| + 4 \leq |E(P_j)| + 2 \leq |E(P_{i_{k+1}})|.$$

Therefore, the sequence of paths $P_{i_1}, P_{i_2}, \cdots$ grow their lengths by least 4 each time, implying that an edge can be bottleneck at most $O(n)$ times. Now since the number of possible edges in the residual network is $2m$, we have the proof.

□