Project Selection Problem

Let V be a set of projects, and there is a profit $p(v)$ associated with project $v \in V$. Note that the profit could be positive or negative. These projects are not independent: to accomplish a certain project, one need to accompolish all its *prerequisite* projects. Such dependency can be modeled as a directed graph $G = (V, E)$, where edge $(u, v) \in E$ represents that v depends on u, i.e., u is one of the prerequisite of v. We say a subset of projects $V_1 \subset V$ satisfies the *prerequisite condition* if for every $v \in V_1$, $(u, v) \in V_1$ implies that $u \in V_1$. Given a directed graph $G = (V, E)$ with profit $p(\cdot)$, the project selection problem seeks subset $V_1 \subset V$ such that V_1 satisfies the prerequisite condition and that the total profit of the selected projects i.e., $\sum_{v \in V_1} p(v)$ is maximized. See Figure 1 for an example.

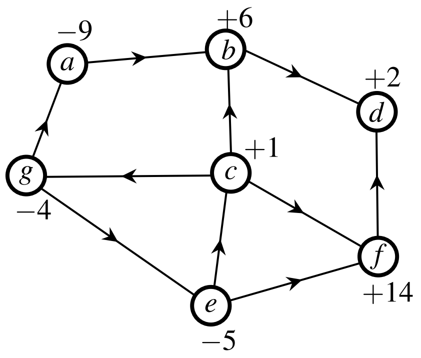


Figure 1: An example of project selection problem. $\{g, c, e\}$ satisfies the prerequisite condition; $\{b, c, d\}$ does not satisfies the prerequisite condition. The optimal solution for this example is $V_1 = \{g, c, e, f\}$, as V_1 satisfies the prerequisite condition and that its total profit $\sum_{v \in V_1} p(v) = 6$ is maximized.

The project selection problem is an abstracted formulation that well models application scenarios that involves making decisions over a set of things that depend on each other. For instance, this formulation can be used to model the election of courses. Specifically, each vertex represents a course and the prerequisites are modeled as edges. Each course is associated with an estimated gain/loss on the overall GPA. Hence, this formulation seeks a subset of courses (satisfying the prerequisite condition, of course) so as to maximize the net gain over one's GPA. Another application scenario is running a company, where one needs to decide over a set of "items" such as hiring people, purchasing equipments, making products, etc. These items clearly depend on each other. For examples, making a product requires hiring people with certain expertise and purchasing certain equipments; a certain equipment can be used by several products, etc. The items can be modeled as vertices and their dependencies be modeled as edges. Items may have an estimated cost (negative profit) or gain (positive profit). Deciding on the items so as to maximize the total profit is clearly another instance of the project selection problem.

We design algorithm to solve this problem. The brute-force approach enumerates all possible subsets (which projects to select), and for each subset one can examine if it satisfies the prerequisite condition, and if it does, calculate its total profit and keep track of the one with maximized total profit. Since there are $2^{|V|}$ possible subsets, this brute-force algorithm certainly runs in exponential time.

We now design a polynomial-time algorithm for above problem, by reducing/transforming it into a network flow problem. Recall that, such reduction involves a 3-step procedure. First, constructing a network (G', s, t, c) based on the input graph G and profit $p(\cdot)$; second, using an existing algorithm for max-flow to obtain the maximum-flow f^* and/or minimum s-t cut (S^*, T^*) of the network G'; third, find the optimal solution V_1, using the found f^* and/or (S^*, T^*).

You may pause now to think about how we apply this framework to solve the maximum-matching problem for bipartite graph. For that problem, we establish that, there exists a one-to-one correspondence between any matching M of the bipartite graph and any (integral) flow f of the constructed network, and that $|M| = |f|$. Hence, finding maximum-matching of the bipartite graph is equivalent to finding maximum-flow of the network. But it is hard to find any connection between the project selection problem with flow. In fact, here we essentially seek a partition of vertices—projects we select and projects we do not select, which is exactly a cut! Hence, for the project selection problem, we instead will establish a one-to-one correspondence between any selection and any s-t cut of the constructed network, and we will also show that minimizing the total profit of the selected projects is equivalent to finding the minimum s-t cut of the network.

We now work out the details of above roadmap. In step #1, we construct the network $(G' = (V', E'), s, t, c)$ as follows. See Figure 2. The directed graph G' incorporates the given graph $G = (V, E)$, not a surprise. We add a source s and a sink t, i.e., $V' = V \cup \{s, t\}$. We connect s to each project with negative profit, and connect each project with positive profit to sink t. Formally, edges in the network is: $E' = E \cup \{(s, v) \mid p(v) < 0\} \cup \{(v, t) \mid p(v) > 0\}$. Now we assign capacities: edges in E, i.e., these in the given graph G, will have a capacity of ∞; edge (s, v) will have a capacity of $-p(v)$; edge (v, t) will have a capacity of $p(v)$. The reasons behind such construction will be revealed when proving the correctness of the algorithm.

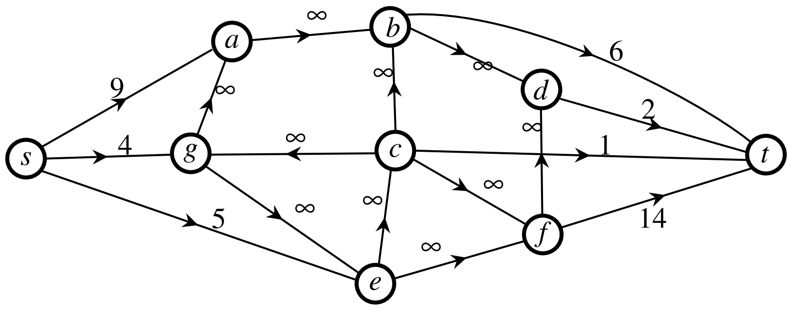


Figure 2: The constructed network (G', s, t, c) for the example in Figure 1.

In step #2, we find a minimum s-t cut (S^*, T^*) of above network (G', s, t, c). In step #3, the algorithm simply returns projects in t-side, i.e., returning $V_1^* = T^* \setminus \{t\}$. See Figure 3.

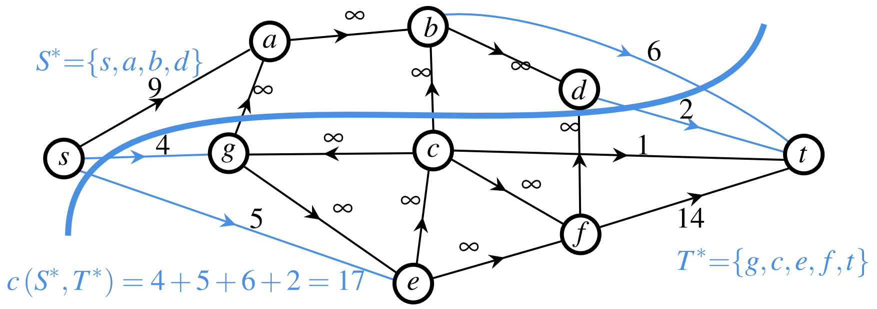
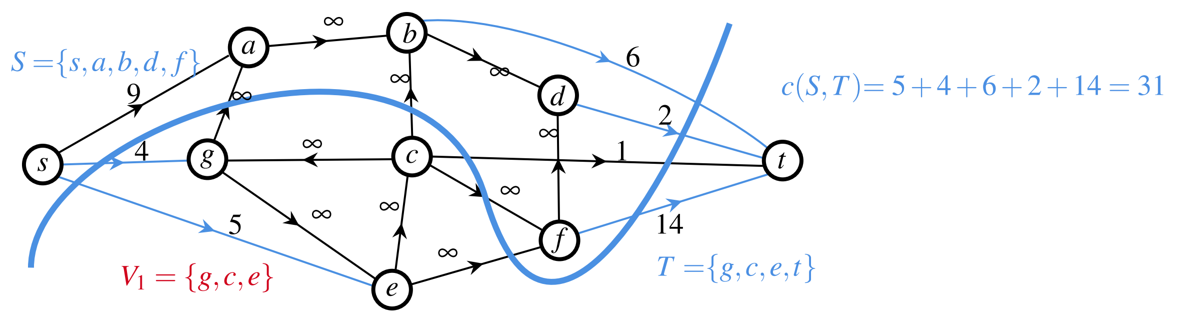


Figure 3: The minimum s-t cut (S^*, T^*) of the network G' in Figure 2. Therefore, for this example, the algorithm returns $V_1^* = T^* \setminus \{t\} = \{g, c, e, f\}$.

We now prove that the above algorithm is correct, i.e., to prove that the returned V_1^* satisfies the prerequisite condition and that $\sum_{v \in V_1^*} p(v)$ is maximized. To achieve this, we establish the one-to-one correspondence between selections and s-t cuts of the built network. Formally, let $V_1 \subset V$ be an arbitrary selection (i.e., a subset of projects). Let $T = V_1 \cup \{t\}$ and $S = (V \setminus V_1) \cup \{s\}$. Clearly, (S, T) forms an s-t cut of the network.

Fact 1. V_1 satisfies the prerequisite condition if and only if $c(S,T) < \infty$.

You may check a couple of examples to fully understand above statement. See Figure 4.



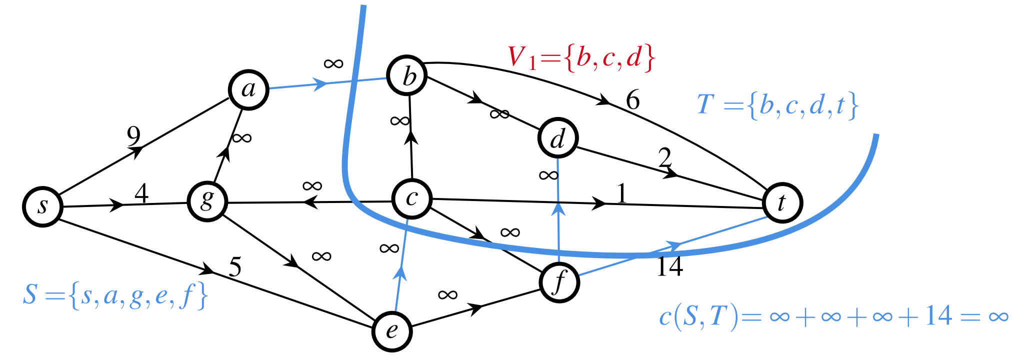


Figure 4: Above: $V_1 = \{g,c,e\}$ which satisfies the prerequisite condition; the corresponding cut capacity $c(S,T) = 31 < \infty$. Below: $V_1 = \{b,c,d\}$ which does not satisfy the prerequisite condition; the corresponding cut capacity $c(S,T) = \infty$. In both examples, cut-edges are marked in blue.

Proof of Fact 1. If V_1 satisfies the prerequisite condition (see the top example of Figure 4), then, by definition, there is no such edge (u,v) where $v \in V_1$ but $u \notin V_1$. Hence, none of edges in E, which have capacity of ∞, belongs to the cut-edges of cut (S,T). So we must have $c(S,T) < \infty$. If V_1 does not satisfy the prerequisite condition (see the bottom example in Figure 4), then there exists edge (u,v) where $v \in V_1$ and $u \notin V_1$. This edge is therefore one of the cut-edges of cut (S,T). Also because the capacity of this edge is ∞, we must have $c(S,T) = \infty$. □

In the network G', s-t cut (S,T) with $c(S,T) < \infty$ exists; for example the s-t cut with just $\{s\}$ on one side. Hence, the minimum s-t cut (S^*,T^*) of the network must have that $c(S^*,T^*) < \infty$, since it is minimum. Combining with above Fact 1, we have the following:

Fact 2. The returned $V_1^* = T^* \setminus \{t\}$ satisfies the prerequisite condition.

It should be clear now why we assign infinite capacity to edges in E when constructing the network G'. This is simply because doing so guarantees that the returned projects satisfies the prerequisite condition. In other words, assigning these edges an infinite capacity *avoids* the cut-edges go through any of them, ensuring that the projects on the t-side satisfies the prerequisite condition.

We now prove that V_1^* maximizes the total profit, by establishing the quantitative relationship between the total profit of any selection and the capacity of the corresponding s-t cut. Let $V_1 \subset V$ be a subset of V that satisfies the prerequisite condition. Let (S,T) be the corresponding s-t cut, where $T = V_1 \cup \{t\}$ and let

$S = (V \setminus V_1) \cup \{s\}$. S and T partition all projects (vertices in V) into two classes. Combining whether a project has a negative or positive profit, all projects are now partitioned into four categories: S_+, S_-, T_+, and T_-, where S_+ consists of projects in S while having positive profit (similar for other three categories). Now the network G' can be sketched in Figure 5. Note that s connects to projects with negative profit, i.e., $S_- \cup T_-$, and t is connected from projects with positive profit, i.e., $S_+ \cup T_+$. Also note that there is no edge from $S_- \cup S_+$ to $T_- \cup T_+$ because we assume that V_1, which is $T_- \cup T_+$, satisfies the prerequisite condition (see Fact 1).

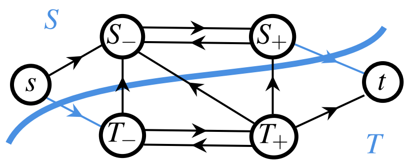


Figure 5: Sketch of the network G' for an s-t cut (S,T).

Now let's establish the connection between capacity of (S,T) and profit of V_1. Note that the cut-edges of (S,T) consists of edges from s to T_- and edges from S_+ to t. We have: $c(S,T) = \sum_{v \in T_-} -p(v) + \sum_{v \in S_+} p(v)$. The total profit of V_1 is: $\sum_{v \in V_1} p(v) = \sum_{v \in T_-} p(v) + \sum_{v \in T_+} p(v)$. We sum up both sides. Note that term $\sum_{v \in T_-} p(v)$ cancels out. We have $c(S,T) + \sum_{v \in V_1} p(v) = \sum_{v \in S_+} p(v) + \sum_{v \in T_+} p(v)$. Note that the right side is exactly the total profit of projects with positive profit, i.e., $\sum_{v \in V : p(v) > 0} p(v)$, which is independent of the V_1 or (S,T). In other words, $c(S,T) + \sum_{v \in V_1} p(v)$ is a constant! Hence, minimizing one term is equivalent to maximizing the other. Since the algorithm finds (S^*, T^*) which minimizes $c(S,T)$, the resulting V_1^* must have the maximized total profit. This is summarized below, which also concludes the proof.

Fact 3. The returned $V_1^* = T^* \setminus \{t\}$ maximizes the total profit.

The time complexity of the algorithm (i.e., the 3 steps) is dominated by the time complexity of calling a max-flow solver. Note that the Ford-Fulkson algorithm does not run in polynomial-time (it in facts runs in pseudo-polynomial-time). But there exists polynomial-time algorithm for maximum-flow, for example the Push-Relabel algorithm. Hence, the algorithm for project selection problem can be solved in polynomial-time as well.