Graphs : Applications of Flow and Cuts[1]

Flows have numerous applications. We sample some.

1 Matchings in Graphs

Imagine there are n workers and n tasks that need to be done. Each worker tells that they can help with at most one task. Furthermore, each worker is qualified to perform only a specified subset of these tasks. Is there a way you can assign these n tasks to workers such that each worker gets at most one task from their specified subset? If not, what is the maximum number of tasks that indeed can be scheduled?

This is a classic application of finding a ***maximum cardinality matching*** in a (bipartite) graph. Recall the concept of matchings and bipartite graphs from your discrete math course. A matching M is a collection of edges which don't share endpoints. A bipartite graph is a graph where the vertex set can be partitioned into two subsets L and R and all edges go from L to R. In the worker-task question above, imagine a bipartite graph where the L vertices correspond to workers, R vertices correspond to tasks, and there is an edge between worker ℓ and task r if the worker ℓ is qualified to do the task r. So, the problem at hand becomes this.

MAXIMUM CARDINALITY BIPARTITE MATCHING
Input: A *bipartite* graph $G = (L \cup R, E)$.
Output: A maximum matching M in G.

We can solve the above problem using a ***reduction*** to maximum flows. Generally speaking, a *reduction* is in fact an algorithm which takes an instance I of a problem Π that we want to solve, and converts it into an instance J of a problem Π' we know how to solve. For this to make sense, the reduction has to be useful. First, given a solution S' to the instance J of the problem we know how to solve, we should be able to obtain a *feasible solution* S for the instance I which we set out to solve. Second, if S' is the optimal solution for J, then S should be the *optimal* solution for I as well.

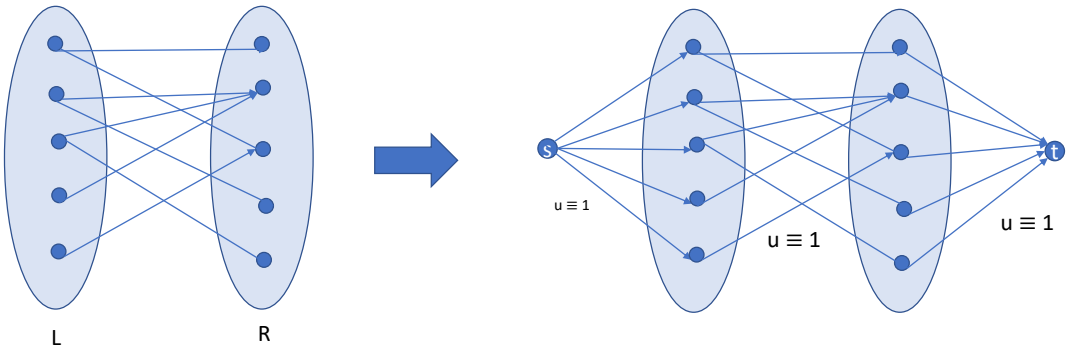
For the matching problem in particular, given the bipartite graph $G = (L \cup R, E)$, we want to first describe the construction of a flow network $\mathcal{N} = (H, s, t, u)$ on which we will solve the maximum s, t-flow problem. Given a maximum flow f in $\mathcal{N}$, we will show how to get a matching M in G. Second, we will reason that this M is the largest sized matching. These three steps will be part of almost all reductions we see in these notes.

Description of Reduction. Construct the network $\mathcal{N} = (H, s, t, u)$ as follows. The figure below shows an illustration.

- The vertices $V(H) = L \cup R \cup s \cup t$ are the vertices of G to which we introduce a source node s and sink node t.
- We now describe the *directed* edges of H. For every $\ell \in L$, we add the directed edge (s, ℓ) to $E(H)$ with $u(s, \ell) = 1$.
- For every $r \in R$, we add the directed edge (r, t) to $E(H)$ with $u(r, t) = 1$.

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 5th Sep, 2022*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

- For all $(\ell, r) \in E(G)$, we add the directed edge (ℓ, r) to $E(H)$ with $u(\ell, r) = 1$.



Feasibility. We solve the maximum s, t-flow problem on $\mathcal{N}$. Since u is integer valued, we actually find an *integer* valued flow $f : E(H) \to \{0, 1\}$ with maximum value $\mathsf{val}(f)$. Note that Ford-Fulkerson itself solves this is $O(nm)$ time. Given such a flow, we simply return the edges $M := \{(\ell, r) \in E(G) \ : \ f(\ell, r) = 1\}$. We claim that this is a matching, and has $|M| = \mathsf{val}(f)$.

Note that if $f(\ell, r) = 1$, then since (s, ℓ) is the only edge coming into ℓ, by flow conservation we get $f(s, \ell) \geq 1$. Since $u(s, \ell) = 1$, we must have $f(s, \ell) = 1$ which implies that there is no other $f(\ell, r') = 1$ for $r' \neq r$. Therefore, we get both properties we need: the edges in M do not share end points, and each edge in M can be associated with a unique edge carrying one unit of flow out of s. Thus, $|M| = \mathsf{val}(f)$.

Optimality. We now prove that the M returned has the largest cardinality. This is via contradiction. Suppose there was a larger matching M'. Then, we use M' to construct a feasible s, t-flow f' in $\mathcal{N}$ with $\mathsf{val}(f') = |M'| > |M| = \mathsf{val}(f)$, and this would contradict that f was a maximum flow.

Let $|M'| = k$, and let $M' = \{(\ell_1, r_1), (\ell_2, r_2), \ldots, (\ell_k, r_k)\}$. Then set $f'(e) = 1$ for all $e \in \{(s, \ell_i), (\ell_i, r_i), (r_i, t) \ : \ 1 \leq i \leq k\}$. This is a feasible s, t-flow with value $|M'|$.

Thus, we have proved the following theorem.

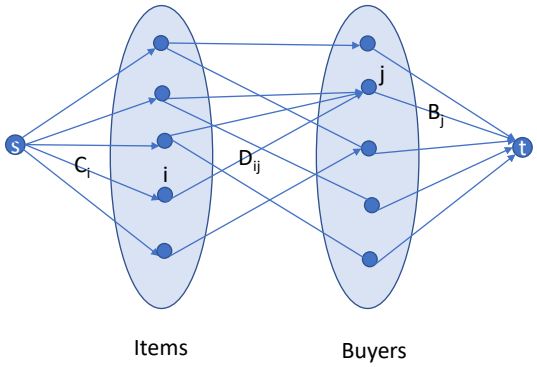
Theorem 1. MAXIMUM CARDINALITY BIPARTITE MATCHING can be solved in $O(nm)$ time.

Many problems, like the task-worker assignment problem, can be reduced to matching itself. If you remember your discrete math, you may recall "Hall's Theorem" which answers when a bipartite graph has a perfect matching (whether all tasks can be performed?). Indeed, the max-flow-min-cut theorem actually proves this almost immediately! See the supplement for this proof. We move on to a few other applications.

2 Slight Generalization : Selling Items

Imagine a market place where the seller has m different kinds of items where they have C_i copies of item i. There are n customers, and customer j comes with a budget of B_j which indicates the total *number* of items they are willing to take home. Furthermore, the seller has an $m \times n$ *demand matrix* D where D_{ij} is a non-negative integer specifying the maximum number of copies of item i customer j is willing to buy. So, if $D_{ij} = 0$ then item i is not of any interest to customer j. How should the sales be made so that the *maximum number* of items are sold?

If you think about it a little, you will see this indeed a generalization of the task-worker problem. B_j was 1 for every j since each worker can do only one task. $D_{ij} = 1$ for the tasks i the worker j is qualified to do, and 0 otherwise. And C_i was 1; each task had only one copy. And indeed, the solution to the sellers problem is a simple generalization of the reduction described in the matching problem. Only the capacities differ. The capacity of source s to item i is C_i; the capacity of buyer j to sink t is B_j, and each (i, j) edge has capacity D_{ij}. We provide the picture below.



It is a recommended exercise for the reader to follow the three steps described for the maximum cardinality matching problem to write a clear reduction to the maximum s, t-flow problem.

3 Generalizing further : A Scheduling Problem

We now show an application to a slightly more complicated problem than the task-worker problem above. You still have n tasks $T_1, \ldots, T_n$ to finish. You have m workers $W_1, \ldots, W_m$ at your disposal. As before, worker W_j is qualified to do a subset of these tasks, but whichever tasks they are qualified for, they can do in 1 hour, and will only do one task at a time. Furthermore, each worker W_j has a calendar of "free" hours at which they can be asked to work. You have access to a small workshop which can hold 2 workers at a time, and you can lease the workshop out on an hourly basis, from 9am to 3pm. Your job is to figure out a *schedule* of which workers should come at which time to the workshop and work on which tasks, so that the maximum number of tasks is completed. Can you figure this out?

Let us look at an illustrative example to better understand the question. Suppose you have 8 tasks, 3 workers, and for simplicity, assume each of these three are qualified to do any of the tasks. Also suppose workers 1 and 2 can only work in the morning (till noon), and worker 3 only in the afternoon (noon to 3pm). Then, one possible schedule is workers W_1 and W_2 work at the workshop from 9 am to 12 pm finishing the first six tasks. Then you can get worker W_3 to finish up the last two tasks from 12pm to 2pm. The problem is to solve the general problem when workers can only do a certain set of jobs, and when they have more general time-constraints.

This problem is also solved by a reduction to a maximum s, t-flow problem but now instead of a bipartite graph, we have a *three* layer graph. This is because we have a tri-partite decision to make: (task, worker, time-slot), and then the capacities encode the various constraints. We begin by describing this reduction.

Description. Given the data, we construct the network $\mathcal{N} = (G, s, t, u)$ as follows.

- $V(G) = T \cup W \cup H \cup \{s, t\}$ where we have a set T of vertices corresponding to the n tasks, a set W of vertices corresponding to the m workers, and a set H of the "hourly time slots" corresponding to the hours the workshop can be leased out. There is also a source vertex s and a sink vertex t.

  We now describe the directed edges in $E(H)$ and their capacities.
- We add an edge (i, j) for $i \in T$ and $j \in W$ iff the worker j is qualified to perform task i. We set $u(i, j) = 1$.
- We add an edge (j, h) for $j \in W$ and $h \in H$ iff worker j is free at the hour time slot h. We set $u(i, j) = 1$.
- We add an edge (s, i) for every $i \in T$ from the source. We set $u(s, i) = 1$.
- Finally, we add an edge (h, t) for every $h \in H$ to the sink. We set $u(h, t) = 2$ to indicate the maximum number of people that can be at the workshop at any time.

Figure 1 shows an illustration of this reduction.

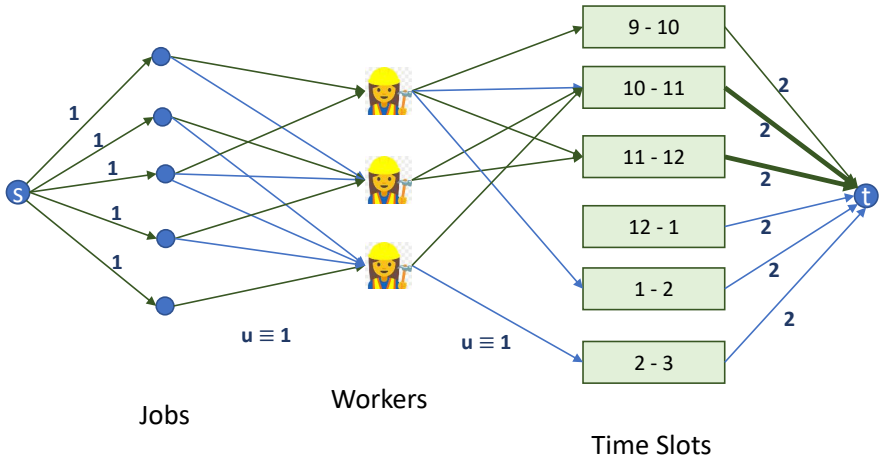


Figure 1: *The green edges shows the assignment of tasks to workers to timeslots. We lease the workshop only in the morning in this solution. Worker* 1 *works on tasks* 1 *and* 3 *from* 9 *am to* 10 *am and then from* 11 *am to* 12 *noon. Worker* 2 *works on tasks* 2 *and* 4 *from* 10 *am to* 12 *noon. Worker* 3 *completes task* 5 *from* 10*am to* 11*am. Thus, from* 10*am to* 12*noon, the workshop has* 2 *workers in it (full capacity).*

Feasibility. We solve the maximum s, t-flow in $\mathcal{N}$, and since the capacities are integer valued, we actually solve to get an integer valued maximum flow f. We now show how to assign jobs to workers to time-slots, such that $\mathsf{val}(f)$ jobs are processed. Note that there are exactly $\mathsf{val}(f)$ edges of the form (s, i) with $f(s, i) = 1$. This uses that the flow on such edges lies in the set $\{0, 1\}$. Pick any such edge (s, i), and we now show an assignment of i to a worker + time-slot. By flow conservation there must exist an edge (i, j) with $f(i, j) > 0$. By integrality, this value must be 1, and thus there is no other j' with $f(i, j') > 0$. We assign i to worker j. Note by construction j is qualified to do the job i. We now repeat the same to obtain edge (j, h) with $f(j, h) = 1$, and again observe that j is free at time slot h. This completes the schedule for job i: it is performed by worker j at time slot s. After this, we decrease $f(s, i)$, $f(i, j)$, $f(j, h)$, and $f(h, t)$ by 1. Note this new flow, f' say, has $\mathsf{val}(f') = \mathsf{val}(f) - 1$ and is also feasible. We repeat this process to find schedules for $\mathsf{val}(f)$ many jobs.

We already seen that no job is given to a qualified worker, and no worker is scheduled on a busy time. To finish the argument that the above schedule is feasible, we need to make sure of two things: one, that no worker j is allocated the same time-slot more than once (since they can only work on one job at a time), and that no more than 2 workers are assigned the same time slot h (since that is the capacity of the workshop). Both are guaranteed by the capacities on the edges. For any worker j and time slot h, the number of times j is assigned h is at most (in fact, it's exactly) the flow $f(j,h) \leq u(j,h) = 1$. Similarly, for any time slot h, whenever it is being used, we decrease the flow $f(h,t)$ by one unit. Therefore, the total number of times the slot h is booked is at most (in fact exactly) $f(h,t) \leq u(h,t) = 2$. Thus we don't overbook, and so the solution we return is a feasible one.

Optimality. To complete the reduction, given any feasible schedule, we show how to construct a feasible s,t-flow f' in $\mathcal{N}$ which has value equaling the number of tasks performed. This will prove that the schedule we find is the best, since otherwise the flow f wasn't a max s,t flow in $\mathcal{N}$. Indeed, whenever task i is performed by job j at time slot h, increase flow $f'(s,i)$, $f'(i,j)$, $f'(j,h)$, and $f'(h,t)$ by 1. Note these are feasible edges in the network. By design, $\mathsf{val}(f')$ equals the number of tasks completed. Since no slot is overbooked, the final flow $f'(h,t)$ is at most 2. Thus, f' is a feasible s,t-flow.

4 Application of Min-Cut: Vertex Cover in Bipartite Graphs

In this section and the next, we look at applications of the minimum s,t-cut problem. More precisely, we will reduce some problems to the minimum s,t-cut problem, and since we know algorithms to find minimum s,t-cuts, we will have algorithms to solve these applications. Our first example is vertex cover.

Suppose we have an undirected graph $G = (V,E)$ and each vertex has a non-negative cost c_v. A subset $S \subseteq V$ of these vertices is a ***vertex cover*** if every edge $(u,v) \in E$ has at least one endpoint in S. That is, S is a set of representative vertices which "hits" every edge. Clearly, $S = V$ is a boring vertex cover. The minimum cost vertex cover problem asks one to find a vertex cover with minimum total cost.

Vertex covers arise in many contexts. For instance, if the graph is a road network and you want surveillance on the roads, then cameras on the vertex cover may suffice to cover every road in the network. Another observation is that the *complement* of a vertex cover (the set of vertices not in S) forms an *independent set*. That is, a collection of vertices not touching each other. The minimum vertex cover problem is equivalent to the maximum independent set problem. This problem also has many applications, one which you have seen — the weighted interval packing problem (see UGP5 to recall this) is a special case of the maximum independent set problem where the vertices are intervals and two intervals have an edge if they intersect. Suffice it to say, both these problems are fundamental graph optimization problems.

Having created this hype, it feels a bit bad to give the bad news: we don't expect fast algorithms to solve either problem on a general graph! However, on *bipartite* graphs, we will now show an algorithm using minimum s,t-cut.

MINIMUM COST VERTEX COVER IN BIPARTITE GRAPHS (MINVC)
Input: A *bipartite* graph $G = (L \cup R, E)$. Costs c_v on vertices
Output: A minimum cost vertex cover $S \subseteq V$ in G.

How should we go about finding this? The connection between MINVC and minimum s,t-min-cut is that both is asking to find some subset of the vertices. And the latter is solving this problem without going over all subsets. So somehow, if we can find a method which takes a MINVC instance and constructs a

network such that (a) capacity of an s, t cut S relates to cost of the vertices in S, and (b) infeasible solutions to the vertex cover problem somehow have "very large" capacity, then we can hope to get this reduction. This is a very general idea; to get from this to the final reduction is something that is sort of an art-form, and one just gets better with practice. I am not going to stress the "how" any more, and just show the reduction.

Description of Reduction. Given a bipartite graph $(L \cup R, E)$ with costs on vertices, we construct a network $N := (H, s, t, u)$ where $V(H) = V(G) \cup \{s, t\}$, we have an edge from s to every vertex of L with capacity $u(s, v) = c_v$. Similarly, we have an edge from every vertex in R to t with capacity $u(v, t) = c_v$. And for every edge (x, y) with $x \in L$ and $y \in R$, we add the same edge to H with capacity $u(x, y) = \infty$. If ∞ bothers you, you can put $M = \sum_{v \in V} c_v + 1$ on each. Figure 2 shows an illustration.

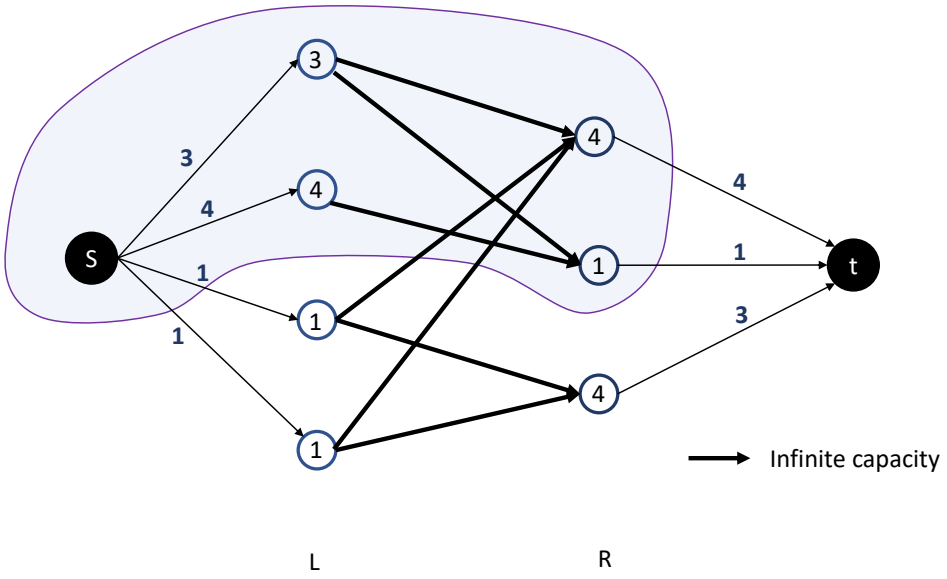


Figure 2: *The costs of the vertices are shown in the circles. The resulting network is shown. A cut set $S \cup s$ is also shown. The set $(L \setminus S) \cup (R \cap S)$ is a vertex cover with cost equaling the capacity of the cut.*

Feasibility. We solve the minimum s, t-cut problem in $\mathcal{N}$. Let $A := S \cup \{s\}$ be the s-side of this cut. Then, we return the set $C = (L \setminus S) \cup (R \cap S)$. We first note that $\texttt{cost}(C) = \mathsf{val}(\partial^+ A)$. We now prove that C is a valid vertex cover. To see this, first we begin with the observation that every edge in $\partial^+ A$ has capacity $\neq \infty$; this is simply because A is the minimum capacity cut, and the cut $\{s\}$ has finite capacity. Therefore, there is no edge from $u \in L \cap S$ to $v \in R \setminus S$. Thus, every edge in the graph G must have either one end point in $L \setminus S$ or $R \cap S$ or both. That is, C is a valid vertex cover.

Optimality. To prove the vertex cover we return has minimum cost, we show that given any vertex cover C', we can obtain an s, t-cut A with $\mathsf{val}(\partial^+ A) = \texttt{cost}(C')$, and this will prove what we want via contradiction. Indeed, given any vertex cover C', the capacity of the cut $A' = (L \setminus C') \cup (R \cap C') \cup s$ is precisely the cost of C. The latter is estabilised using the fact that C' is a vertex cover and so there are no infinite cost edges crossing the cut.

Remark: *We should remark that the above reduction only works when the costs c_v are non-negative as the capacities on edges in the max-flow-min-cut problem* ***needs*** *to be non-negative. However, negative costs do not cause a problem for the following reason: if $c_v < 0$, then v will be in every minimum cost vertex cover because a superset of a vertex cover is also a vertex cover. Therefore, we pick all the vertices N which have negative cost and remove them* and *all edges incident on them (even the ones with only one end-point in N as they have been covered) and apply the above algorithm in what remains. Note that bipartite graphs remain bipartite on deletion, and therefore we are ok.*

5 A Project Selection Problem

In this problem we are given a *directed acyclic graph* $G = (V, E)$. Each node v_i corresponds to a project, and has an associated value p_i. This value could be *positive*, that is, completing this project gives you p_i units of revenue, or it could be *negative*, completing this project leads to a loss of p_i units.

The edges (v_i, v_j) are *precedence constraints* – to perform a project v_j one must complete the project v_i as well. This is similar to a problem in your PSet. Indeed, if you think of these projects as classes you need to take in Dartmouth. Then you can imagine one vertex titled "Major" which has some value (hopefully, significantly positive for everyone). However, there is a node titled "CS 31" which points to it. And it may (or hopefully may not) have negative value. However, to complete the "Major" project, you need to complete the "CS 31" project.

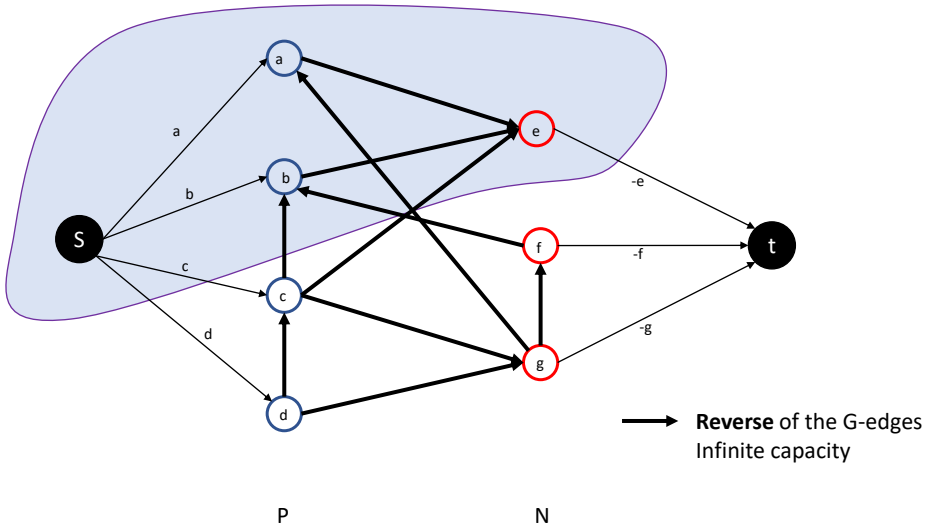
The objective of the problem is to select a subset of projects $S \subseteq V$ to maximize the total value $\sum_{i \in S} p_i$. The constraint is: if a vertex $v \in S$, then for all edges (u, v), the vertex $u \in S$ as well. Such a set is called ***valid***. How will we attack this problem? We will try something similar to what we did for the minimum vertex cover problem. We want to construct a network such that somehow the *minimum* s, t-cut problem leads us the the *maximum* valid subset problem for project selection. Note that somehow we need to solve the maximization problem using a minimization problem. Interesting!

Let us begin with let us figure out *trivial* lower bounds and upper bounds on the maximum value that can be obtained. Note, that we could pick no project, and this gives us a total value of 0. On the other hand, we could only pick the projects which give us *positive* reward. Of course this set may not be feasible, but we cannot get more than the total profit of the positive rewards. This benchmark we call $\Theta := \sum_{v \in V : p_v > 0} p_v$. Next let us describe the *reduction* to the minimum s, t-cut problem.

Description of Reduction. Given the input above, we describe a flow network $\mathcal{N} = (H, s, t, u)$.

- $V(H) = V(G) \cup \{s, t\}$, that is, the vertex set is the original vertices plus a source and a sink.
  To describe the edge set, we partition $V(G)$ into two groups: $P \cup N$ where $P := \{i \in V(G) : p_i > 0\}$ are the projects which give positive value, and $N := \{i \in V(G) : p_i \leq 0\}$ are the rest.
- For each vertex $v_i \in P$, we add an edge (s, v_i) of capacity p_i.
- For each vertex $v_j \in N$, we add an edge (v_j, t) of capacity $-p_j$. Note that these capacities are non-negative since $p_j \leq 0$.
- Finally, for every edge (v_i, v_j) in the graph G, we add the ***reverse*** edge (v_j, v_i) in the graph H. We assign a capacity ∞ to each such edge.

See the figure below as illustration.



Feasibility. Now, our algorithm for the project selection problem is the following. We obtain a minimum s,t-cut for the network $\mathcal{N}$. If this minimum cut is $\partial^+ A$ where $A = S \cup \{s\}$ is the s-side of the cut, then we return S as the set of projects.

Let's prove that S is a valid solution. Suppose not. Suppose there is a project $v \in S$ and another project $u \notin S$, such that $(u, v) \in G$. That is, we chose v but not its pre-req. But in that case $(v, u) \in E(H)$ and in fact, it is in $\partial^+ A$. That would make the capacity of $u(\partial^+ A) = \infty$ contradicting that A was a minimum s,t-cut. Thus, S is a valid set.

Optimality. We begin by noting that the value of the set S is precisely $\Theta - u(\partial^+ A)$, where recall $\Theta = \sum_{v \in P} p_v$. This follows three things and some algebra: (a) $\mathsf{val}(S) = \sum_{v \in P \cap S} p_v + \sum_{v \in N \cap S} p_v$, (b) $u(\partial^+ A) = \sum_{v \in P \setminus S} p_v + \sum_{v \in N \cap S} (-p_v)$, and (c) $\Theta = \sum_{v \in P \cap S} p_v + \sum_{v \in P \setminus S} p_v$. For instance, in the figure above, the capacity of the cut is $(c{+}d{+}(-e))$, and $\Theta = a{+}b{+}c{+}d$. Note that $\Theta - u(\partial^+ A) = a{+}b{+}e$ which is precisely the total profit in S.

The proof completes by noting that if $S' \subseteq V$ is any valid solution for the project selection problem. Then $A \cup s$ induces an s,t-cut of capacity $u(\partial^+ S) = \Theta - \mathsf{val}(A)$. Since A is valid, there is no edge from $v \in A$ to $u \notin A$; thus the capacity of $\partial^+(A \cup s)$ is precisely the capacity of the edges from s to $u \in P \setminus A$ and $N \cap A$ to t. The capacity of these edges precisely is $\Theta - \mathsf{val}(A)$.