Graphs : Applications of Flow and Cuts[1]

Flows have numerous applications. We sample some.

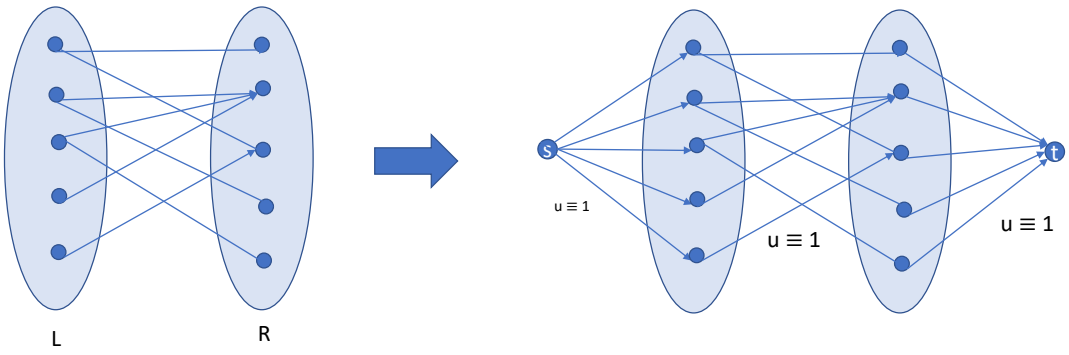
1 Matchings in Graphs

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 CS30 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. Given the graph G, we construct a flow network $\mathcal{N} = (G, s, t, u)$ as follows. We introduce a source node s and sink node t. We add edge (s, ℓ) for all $\ell \in L$ and add edge (r, t) for all $r \in R$. We set $u(e) = 1$ for all these edges and the edges $e \in G$. The figure below shows an illustration.



Lemma 1. The value of the maximum flow in $\mathcal{N}$ is the size of the maximum cardinality matching in G.

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 19th Mar, 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!

Proof. Let M be the maximum matching in G. Let $|M| = k$, and let $M = \{(\ell_1, r_1), (\ell_2, r_2), \ldots, (\ell_k, r_k)\}$. We can send a flow of value k in $\mathcal{N}$ by sending them along the paths (s, ℓ_i, r_i, t).

Conversely, if there is a flow of value k in $\mathcal{N}$, then using integrality of flow we may assume this flow has $f(e) \in \{0, 1\}$ on all edges. Thus, we have k edge disjoint paths from s to t in $\mathcal{N}$. Focusing on the edges of G in these paths, we get a matching of size k. □

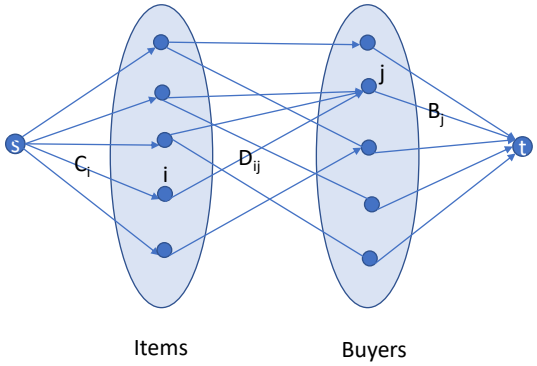
Theorem 1. MAXIMUM CARDINALITY BIPARTITE MATCHING can be solved in $O(nm)$ time.

Many problems, like the task-worker assignment problem, can be reduced to matching itself. If you remember your CS30, 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.



Lemma 2. The value of the maximum flow in the above network is precisely the maximum number of items the seller can sell.

Proof. Consider the best way in which a seller can sell their items. We show that these sales correspond to a flow in the network. Indeed, if any copy of item i is sold to buyer j, then we send flow $f(s,i)+=1$, $f(i,j)+=1$, and, $f(j,t)+=1$ on the three edges. The flow is feasible since (a) there are at most C_i copies of item i sold, (b) no more than D_{ij} copies of item i were sold to buyer j, and (c) buyer j buys at most B_j items in total. The value of the flow is precisely the number of items sold.

Conversely, given any maximum flow, *which is integral* since all the data is integetal, we can read out how to make our sales. □

3 Generalizing further : A Scheduling Problem

We now show an application to a 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 at a time. Furthermore, each worker W_j has a calendar of "free" hours at which time the 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 me give an illustrative example. Suppose you have 8 tasks, 3 workers, and for simplicity, assume each of these three are qualified to 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 time-constraints. Can you see how this is a flow problem?

This 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. There is a set T of vertices corresponding to the n tasks, a set W of vertices corresponding to the m workers, and a set of six "hourly time slots" corresponding to the hours the workshop can be leased out. We have an edge from task vertex i to worker vertex j if and only if W_j is qualified to perform task T_i. We have an edge from worker vertex j to time slot s if W_j is free to work in the time slot s. All these edges have capacity 1. We have a source and sink vertex. There is an edge from source vertex s to every task vertex i with capacity 1. There is an edge from every time slot s to a sink vertex t. The capacity of each of these vertices is 2 indicating the number of people can be at the workshop at any time. Figure 1 shows an illustration.

We claim that there is a way to lease out the workshop and assign tasks to workers, and schedule them in the workshop if and only if there is a feasible flow of value n (number of tasks) in the above network. Given a schedule, we can send a flow. Given a maximum flow, again using integrality of the flow since data is integer valued, we can read out an assignment. See Figure 1 for an illustration of this as well.

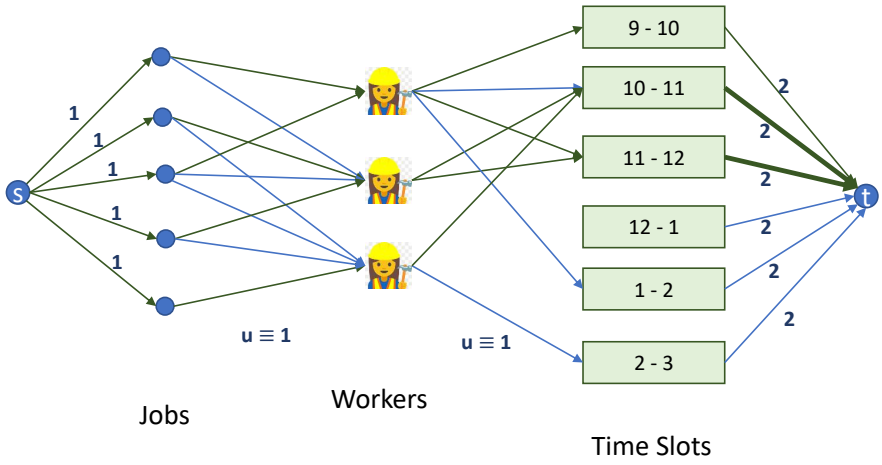


Figure 1: The green edges shows the assignment of tasks to workers to timeslots. We lease the workshop only in the morning in this solution. Worker 1 works on tasks 1 and 3 from 9 am to 10 am and then from 11 am to 12 noon. Worker 2 works on tasks 2 and 4 from 10 am to 12 noon. Worker 3 completes task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

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

In this section and the next, we look at how the minimum s, t-cut can have applications. 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.

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 $C = \sum_{v \in V} c_v + 1$ on each. Figure 2 shows an illustration. The reduction follows from the following

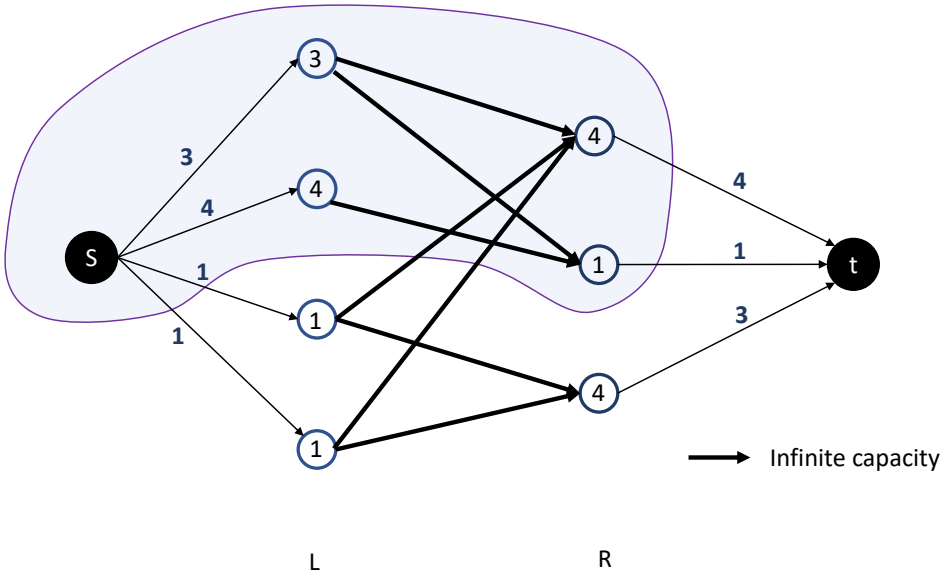


Figure 2: The costs of the vertices are shown in the circles. The resulting network is shown. A cut set $S \cup s$ is also shown. The set $(L \setminus S) \cup (R \cap S)$ is a vertex cover with cost equaling the capacity of the cut.

claim.

Lemma 3. Let $(S \cup s)$ be an s, t-cut in N with *finite* capacity. Then the set $C = (L \setminus S) \cup (R \cap S)$ is a valid vertex cover whose cost is precisely the capacity of the s, t-cut $(S \cup s)$. Conversely, given any vertex cover C, the capacity of the cut $S = (L \setminus C) \cup (R \cap C) \cup s$ is precisely the cost of C.

Proof. If $(S \cup s)$ is a finite capacity cut, then there *cannot* be any 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, $C = (L \setminus S) \cup (R \cap S)$ is a vertex cover. Furthermore, the capacity of the cut is precisely $\sum_{v \in L \setminus S} u(s, v) + \sum_{v \in R \cap S} u(v, t)$. This is precisely the cost of C. Similarly, given a vertex cover C, one can construct S as stated in the lemma. The fact that it is a vertex cover implies there are no infinite cost edges crossing the cut. And its capacity, like above, is precisely the cost. □

This implies that the minimum cost vertex cover in G can be found by computing the minimum capacity s, t cut in N.

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.

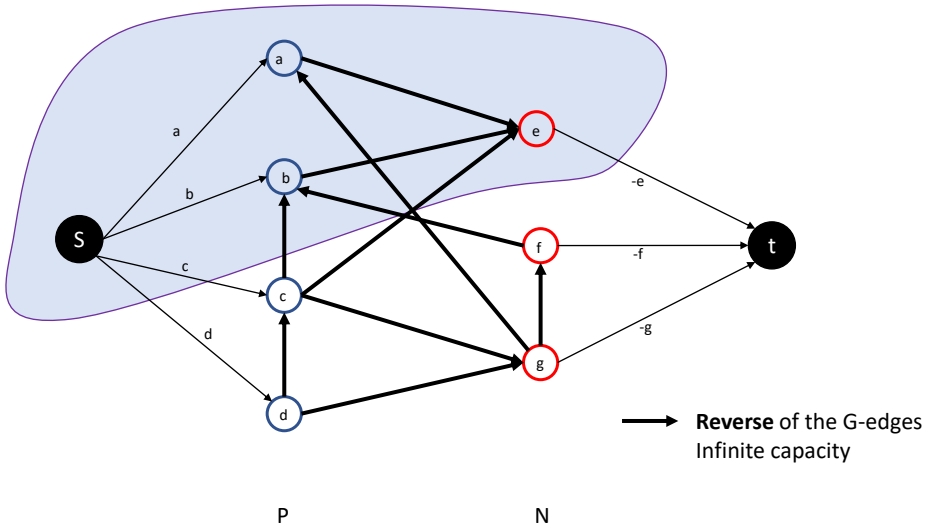
The edges (v_i, v_j) are *precedence constraints* – to perform a project v_j one must complete the project v_i as well. This is similar to a problem in your PSet. Indeed, if you think of these projects as classes you need to take in Dartmouth. Then you can imagine one vertex titled "Major" which has 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. Given the input above, we describe a flow network $N = (H, s, t, u)$. $H = (V \cup s \cup t)$, that is, we have the original vertices plus a source and a sink. We partition V into two groups: $P \subseteq V$ with $p_i > 0$, that is, the projects which give positive value. We add an edge (s, v_i) for each $v \in P$ of capacity p_i. The remaining vertices N, for not positive, have $p_i \leq 0$. For each vertex $v_j \in N$, we add an edge (v_j, t) of capacity $-p_j$. Note that the capacities are non-negative.

Next, 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.



Now, our algorithm for the project selection problem is the following. We obtain a minimum s, t-cut for the network N. If this minimum cut is $\partial^+ S$, then we choose $S \setminus s$ as the set of projects. We claim this is the best solution. This follows from the two claims below.

Claim 1. If $u(\partial^+ S) \neq \infty$, then the set $S \setminus s$ is a *valid* set. The total value of the set $S \setminus s$ is precisely $\Theta - u(\partial^+ S)$.

Proof. Let's first prove that $S \setminus s$ is valid. Suppose not. Suppose there is a project $v \in S \setminus s$ and another project $u \notin S$, such that $(u, v) \in G$. But then, $(v, u) \in H$. And that would make the capacity of $u(\partial^+ S) = \infty$. Thus, $S \setminus s$ is a valid set.

The total value of the set is the $\sum_{v \in S \cap P} p_v + \sum_{v \in S \cap N} p_v$. Note that for every $v \in S \cap N$, we have $\sum_{v \in S \cap N}(-p_v)$ equal the capacity of the edges from S to t. Thus, $\sum_{v \in S \cap N} p_v$ equals the negative of the capacity of the edges from S to t. Note that $\sum_{v \in S \cap P} p_v = \sum_{v \in P} p_v - \sum_{v \notin P \setminus S} p_v = \Theta$ minus the total capacity of edges from the source s to outside S. Thus, $\sum_{v \in S \cap P} p_v + \sum_{v \in S \cap N} p_v = \Theta - u(\partial^+ S)$.

For instance, in the figure above, the capacity of the cut is $(c + d + (-e))$, and $\Theta = a + b + c + d$. Note that $\Theta - u(\partial^+ S) = a + b + e$ which is precisely the total profit in S. □

Claim 2. Let $A \subseteq V$ be any valid solution for the project selection problem. Then $A \cup s$ induces an s, t-cut of capacity $u(\partial^+ S) = \Theta - \mathsf{val}(A)$.

Proof. 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 $\Theta - \mathsf{val}(A)$. □