CS 341 - Algorithms, Winter 2023. University of Waterloo

Lecture 16: Applications of Max-Flow Min-Cut

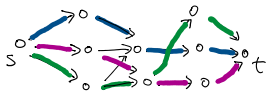
We see two basic applications and two fun applications of maximum flows and minimum cuts.

Disjoint Paths

As discussed in L15, the maximum edge-disjoint s-t path problem can be reduce to maximum s-t flow.

Input: A directed graph $G=(V,E)$, two vertices $s,t \in V$.

Output: A maximum-sized subset of edge-disjoint s-t paths



To recall the reduction, given a directed graph, simply set the capacity of every edge to be one.

Claim There are k edge-disjoint s-t paths if and only if there is an s-t flow of value k.

Proof On one hand, if $P_1, \ldots, P_k$ are edge-disjoint s-t paths, just set the flow on each edge on these paths to be one, and it is easy to check that this gives a flow of value k.

On the other hand, if f is an s-t flow of value k, then we can apply the flow decomposition lemma in L15 to obtain k edge-disjoint s-t paths. (This is also discussed in L15.) □

Check that the Ford-Fulkerson algorithm thus provides an $O(|V|\cdot|E|)$-time algorithm for edge-disjoint s-t paths.

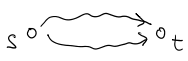
Also check that the max-flow min-cut theorem implies the following min-max theorem.

Theorem The maximum number of edge-disjoint s-t paths is equal to the minimum number of edges whose removal disconnects t from s (i.e. no paths from s to t after removal).

Vertex Disjoint Paths

What if we want s-t paths that are vertex disjoint?

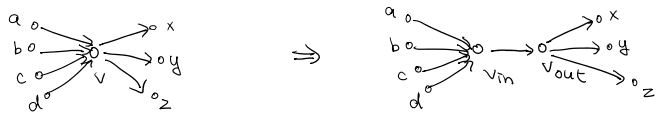
Two s-t paths P_1 and P_2 are called (internally) vertex-disjoint if they only share the vertices s and t, but not any other vertex.



It turns out that this problem can be reduced to maximum edge-disjoint s-t paths.

The idea is to create an edge to "imitate" a vertex.

Reduction: For each vertex $v \in V-s-t$, replace it by a two-vertex structure as follows:



Claim There are k vertex-disjoint s-t paths in the original graph if and only if there are k edge-disjoint s-t paths in the new graph.

This gives a $O(|V| \cdot |E|)$-time algorithm for the maximum vertex-disjoint s-t path problem, and the max-flow min-cut theorem implies the following min-max theorem.

Theorem: The maximum number of vertex-disjoint s-t paths is equal to the minimum number of vertices whose removal disconnects t from s (i.e. no paths from s to t after removal).

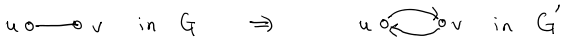
We leave the (straightforward) details to the reader.

Undirected Graphs

We could also ask the maximum edge-disjoint s-t path problem for undirected graphs.

This problem can also be reduced to the maximum edge disjoint s-t path problem for directed graphs.

The reduction is simple: for each edge uv in G, have both $u \to v$ and $v \to u$ in the directed graph G'.



Claim: There are k edge-disjoint s-t paths in G iff there are k edge-disjoint s-t paths in G'.

You may worry that both $u \to v$ and $v \to u$ are used in the edge disjoint paths in G' and thus the corresponding undirected paths are not edge-disjoint, but we can argue that there is a solution in G' such that no "anti-parallel" edges (i.e. $u \to v$ and $v \to u$) are both used. We leave it to the reader to figure out.

So, we also have a polynomial time algorithm and a min-max theorem for undirected edge-disjoint s-t paths.

What about vertex-disjoint s-t paths in undirected graphs? We leave it as an exercise.

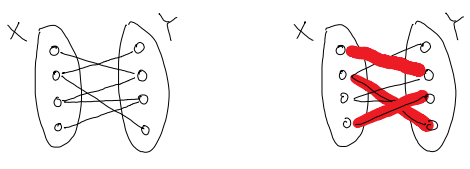
Exercise: Reduce maximum vertex-disjoint s-t paths in undirected graphs to a previous problem. Obtain a polynomial time algorithm and derive a min-max theorem for this problem.

Bipartite Matching

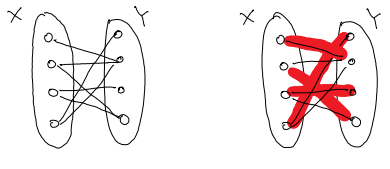
The bipartite matching problem is an important problem both in theory and in practice.

Input: A bipartite graph $G = (X, Y, E)$.

Output: A maximum cardinality subset of edges that are vertex disjoint.



maximum matching



perfect matching

A subset of edges $M \subseteq E$ is called a matching if edges in M are pairwise vertex disjoint, or in other words, no two edges in M share a vertex.

We are interested in finding a maximum matching, a matching with the maximum number of edges.

A matching is called a perfect matching if every vertex in the graph is matched.

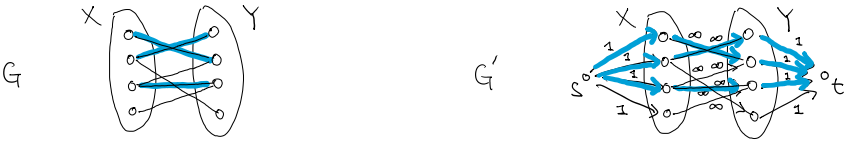
Obviously, a perfect matching is the best that we can hope for for the maximum matching problem.

Reduction to Maximum Flow

The maximum bipartite matching problem can be reduced to the maximum flow problem.

Given a bipartite graph $G = (X, Y, E)$, we construct a directed graph G' as follows:

- The vertex set of G' is $\{s\} \cup X \cup Y \cup \{t\}$, where s and t are two new vertices.
- All the edges in E appear in G', but pointing from X to Y, with edge capacity being infinity.
  There is an edge from s to every vertex in X, with edge capacity one.
  There is an edge from every vertex in Y to t, with edge capacity one.



Claim There is a matching of size k in G iff there is an s-t flow of value k in G'.

Proof It should be clear from the picture that a matching of size k gives a flow of value k.

In the other direction, given an integer flow of value k (which is guaranteed by Ford-Fulkerson's algorithm), we apply the flow decomposition lemma in LIS to obtain k s-t paths of capacity one.

Note that each s-t path in G' has exactly 3 edges by our construction of G'.

Since each edge from s and to t has capacity one, these k paths must be (internally) vertex disjoint.

Removing the edges from s and the edges to t from these paths gives us k vertex disjoint edges, a matching of size k in G.

Through this reduction, the Ford-Fulkerson algorithm gives a $O(|V| \cdot |E|)$-time algorithm for bipartite matching.

Remark: You may wonder why we assign edge capacity infinity to the original edges - but not edge capacity one. It will also work for this reduction, but infinity capacity will be more convenient later.

Minimum Vertex Cover

What should be the min-max theorem for bipartite matching? Consider the vertex cover problem.

Given a graph $G = (V, E)$, a subset of vertices $S \subseteq V$ is called a vertex cover

if for every edge $uv \in E$, $\{u,v\} \cap S \neq \phi$.

In words, S is a vertex cover if S intersects every edge.

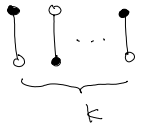
Input: A bipartite graph $G=(V,E)$

Output: A vertex cover of minimum cardinality.

Some examples:

The vertex cover problem may seem to be unrelated to the matching problem, but they are actually "dual" of each other on bipartite graphs, in a precise and meaningful way.

To start to see their connection, first we observe that if there is a matching with k edges, then any vertex cover must have at least k vertices.



The reason is simple: since the k matching edges are vertex disjoint, we must use k distinct vertices just to cover these k edges, and so any vertex cover must have at least k vertices.

Therefore, the size of a maximum matching is a lower bound on the size of a minimum vertex cover.

Surprisingly, this is always a tight lower bound on bipartite graphs.

Put it in another way, having a large matching is the only reason that we need a large vertex cover.

Theorem (König) In a bipartite graph, the maximum size of a matching is equal to the minimum size of a vertex cover.

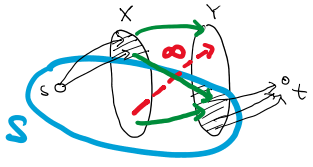
Proof We prove König's theorem using the max-flow min-cut theorem.

Let the maximum size of a bipartite matching in G be k. We would like to show a vertex cover of size k.

From the reduction above, the maximum s-t flow in G' above is of value k.

By the max-flow min-cut theorem, there is an s-t cut S in G' of capacity k, i.e. $c^{out}(S) = k$.

Consider such a set S, with $s \in S$ and $t \notin S$.



There can be at most k vertices in $(X-S) \cup (S \cap Y)$,

because s has an edge of capacity one to each vertex in $X-S$,

and every vertex in $Y \cap S$ has an edge of capacity one to t.

These edges are in $\delta^{out}(S)$ and so $k = c^{out}(S) \geq |X-S| + |S \cap Y|$.

We claim that the vertices in $(X-S) \cup (S \cap Y)$ is a vertex cover of G, with size at most k.

To prove this claim, we just need to show that there are no edges with one endpoint in $S \cap X$ and another endpoint in $Y-S$, but this is true because each such edge would be a directed edge in $\delta^{out}(S)$ in G' with infinity capacity, contradicting that $c^{out}(S) = k$.

another endpoint in $Y-S$, but this is true because each such edge would be a directed edge in $\delta^{out}(S)$ in G' with infinity capacity, contradicting that $c^{out}(S) = k$.

This concludes that $(X-S) \cup (S \cap Y)$ is a vertex cover in G of size k, proving König's theorem. □

Note that this proof provides a $O(|V| \cdot |E|)$-time algorithm to solve the vertex cover problem on bipartite graphs.

Good Characterization (Discussions)

König's theorem is a nice example of a graph-theoretic characterization.

To see this, imagine that you work for a company and your boss asks you to find a maximum matching say to assign some jobs to the employees.

If your algorithm finds a perfect matching, then your boss would be happy.

But suppose there is no perfect matching in the graph. how could you convince your boss about the non-existen

Your boss may just think that you are incompetent and other smarter people could find a better assignment.

A convincing way is to show your boss a vertex cover of size $< \frac{n}{2}$, and explain that if there is a perfect matching such a vertex cover could not exist.

These min-max theorems are some of the most beautiful results in combinatorial optimization, providing succinct "proofs" for both the YES-case (a large matching) and the No-case (a small vertex cover).

They show the non-existence of a solution by the existence of a simple obstruction.

Remark: To put the above discussion into perspective, consider the dynamic programming algorithms.

Even though we could solve the problems (such as optimal binary search tree) in polynomial time, we don't have such a nice succinct characterization for the No-cases.

If your boss asks why there is no better solution, it would be quite difficult to explain (e.g. we search for all possible solutions using this recurrence and our is an optimal solution).

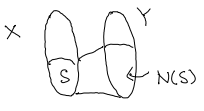
The "proof" is still succinct in the sense that it is a polynomial sized table. but it is not nearly as elegant as the proofs provided by the min-max theorems.

Hall's Theorem characterizes when a bipartite graph $G = (X, Y; E)$ has a perfect matching or not.

Without loss of generality, we assume that $|X| = |Y|$, as otherwise there is no perfect matching.

Theorem (Hall) A bipartite graph $G = (X, Y; E)$ with $|X| = |Y|$ has a perfect matching if and only if for every subset $S \subseteq X$, it holds that $|N(S)| \geq |S|$ where $N(S)$ is the neighbor set of S in Y.

(In words, every subset $S \subseteq X$ has at least $|S|$ neighbors in Y.)



Exercise : Derive Hall's theorem from König's theorem.

Hall's theorem is an important result and has many applications in graph theory.

We show one corollary, which may be used in homework.

Corollary Every d-regular bipartite graph $G=(X,Y;E)$ has a perfect matching.

Proof There are $d|X|$ edges in the graph. Note that $|X|=|Y|$ as G is regular.

Any subset of vertices S with $|S| \le |X|-1$ can cover at most $d(|X|-1)$ edges.

So, any vertex cover needs at least $|X|$ vertices.

By König's theorem, there is a matching of size at least $|X|=|Y|$, hence a perfect matching. □

Duality: (Discussion) Why vertex cover is the "dual" problem of matching? How could we come up with it?

Actually, there is a systematic way to define the dual problem of an optimization problem, through the use of linear programming.

Most combinatorial optimization problems can be solved in the general framework of linear programming.

This is one of the most, if not the most, powerful algorithmic framework for polynomial time computation.

The many beautiful min-max theorems can also be systematically derived from linear programming duality.

Unfortunately, we won't learn about linear programming in this course; see [kT, DPV, CLRS] for more.

There are also various courses in the C&O department on these topics.

Baseball / Basketball League Winner (optional) [kT 7.12]

Suppose you are a fan of Toronto Blue Jays or Toronto Raptors or Waterloo Warriors.

They are playing in the regular season. They are doing okay but not at the top of the table.

You wonder whether it is still (mathematically) possible for them to finish on top in the end of the season

Input: The current standing / table, and the remaining schedule.

Output: Whether it is possible that your team can win the league.

The feature of the baseball / basketball league is that there is only win or lose (i.e. no draw as in football).

Suppose your team has currently won W games.

First, it is obvious that we assume that they will win all their remaining games, for a total of W^* games.

Then, we want to know whether there is a scenario that every other team will win less than W^* games.

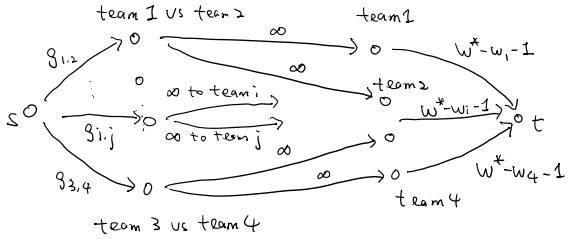
Let the other teams be $\{1,2,\ldots,n\}$ and currently team i has won w_i games.

We have the remaining schedule, showing that there are still g_{ij} games to be played between team i and team j.

How do we determine if there is still a scenario that your team can still win the league?

Reduction: This problem can be reduced to a maximum s-t flow problem.

The idea is to "assign" the winner of each remaining game so that no team will win more that $w^* - w_i - 1$ games.



The vertex set is $\{s\} \cup P \cup T \cup \{t\}$ where P has a vertex for each pair of teams and T has a vertex for each team.

There is an edge with capacity $g_{i,j}$ to each vertex $p_{i,j} \in P$, as there are still $g_{i,j}$ games between team i and j.

For $p_{i,j} \in P$, there is an edge from $p_{i,j}$ to $t_i \in T$ and from $p_{i,j}$ to $t_j \in T$, so that each game between team i and team j will be sent to either team i or team j, whoever is the winner of that game.

For each $t_i \in T$, there is an edge from t_i to the sink t with capacity $w^* - w_i - 1$, as we want team i to win at most $w^* - w_i - 1$ games.

Let $g = \sum_{i,j} g_{i,j}$. We want all these games to be played, with one winner for each game, so that team i wins at most $w^* - w_i - 1$ games for all i (and thus our favorite team with w^* wins is the champion).

With this intention in mind, it is not difficult to check the following claim.

Claim Our favorite team can still win the league iff there is an s-t flow of value g in the graph.

Remark: It is an NP-hard problem to determine if your favorite football team can still win the league (i.e. winning team gets 3 points, each team gets one point in a draw).

Project Selection (optional) [KT 7.11]

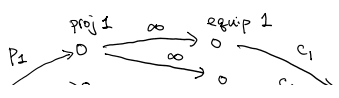
There are n projects available, completing each will give us profit p_i dollars.

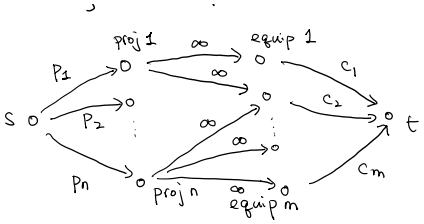
To do each project, however, we need to buy a subset of equipments, each will cost c_j dollars.

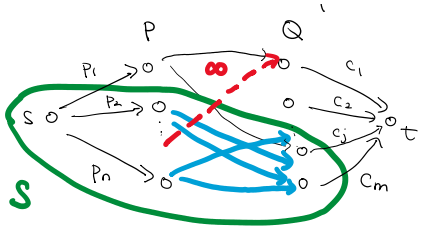
An equipment, once bought, can be used for multiple projects that require this equipment.

Now, the question is, what projects we should do and what equipments we should buy so that we can maximize the net profit, i.e. total profit from projects minus total costs from equipments.

Reduction: Interesting, this problem can be reduced to the minimum s-t cut problem.







The vertex set is $\{s\} \cup P \cup Q \cup \{t\}$. where P has a vertex for each project and Q has a vertex for each equipment.

There is an edge from s to project i with capacity p_i, and an edge from equipment j to t with capacity c_j.

Also, there is an edge with capacity infinity from project i to equipment j if it is required for project i.

Let $p = \sum_{i=1}^{n} p_i$. Clearly, this is the best possible net profit that we can hope to make, earning the profit of every project without buying any equipment.

Now, we want to minimize the difference to this ideal profit p, by minimizing the cost of the equipments that we need to buy and the profit of the projects that we need to miss.

Consider a minimum s-t cut S with $c^{out}(S) = k$.

We claim that it is possible to make a net profit of $p-k$.

Buy all the equipments in $S \cap Q$ and take all the projects in $S \cap P$.

The key point is that it is a feasible solution, as there are no edges from $S \cap P$ to $Q - S$, otherwise this is an outgoing edge of S with infinity capacity, contradicting $c^{out}(S) = k$.

So, all the equipments that the projects in $S \cap P$ required are in $S \cap Q$.

The net profit of this solution is $\sum_{i \in S \cap P} p_i - \sum_{j \in S \cap Q} c_j = p - \sum_{i \in P-S} p_i - \sum_{j \in S \cap Q} c_j = p - c^{out}(S) = p-k$.

In other words, the minimum s-t cut S tells us how to minimize the cost of the equipments that we need to buy (equipments in $S \cap Q$) plus the profit of the projects that we need to miss (projects in $P-S$).

On the other hand, a feasible solution with net profit $p-k$ gives us an s-t cut with capacity k, so we have the following conclusion.

Claim The maximum net profit is $p-k$ if and only if the capacity of a minimum s-t cut is k.

References: [KT 7.5, 7.6, 7.11, 7.12]

There are many more useful and interesting applications of max-flow min-cut.

You can learn more from the CSO network flow course.

You can also learn more about linear programing from CSO, which is the general underlying framework of the local search method as well as the duality theory behind the min-max theorems.