Dual problems and linear programming

Min-cut-max-flow

The min-cut max-flow theorem is a powerful culmination of various foundational results in graph theory: Hall's, König's, and Menger's Theorems all are really special cases of the min-cut max-flow idea.
We follow the proof presented in 6.2 in Diestel [1], with less precision. We will consider directed graphs for this topics: simple graphs in which every edge has a direction. We denote xy for an edge $x \to y$. The direction of the edge does not imply anything about which direction flows are possible, it is only a label to help distinguish the directions. Furthermore, suppose that our graph has two special vertices, a *source* s, and a *sink* t.

We define a *flow* as a function $f : E(G) \to \mathbb{N}$. We let $f(v, S) = \sum_{w \in S} f(v, w)$ for any $v \in V$ and $S \subseteq V$. Furthermore, we let the *capacity* be a function $c : E(G) \to \mathbb{N}_{\geq 1}$.

- $f(x, y) \leq c(x, y)$ for all $(x, y) \in E(G)$,
- $f(x, y) = -f(y, x)$ for all $(x, y) \in E(G)$,
- $f(V, s) = f(t, V) = 0$,
- $f(V, v) = f(v, V)$ for all $v \in V(G) \setminus \{s, t\}$.

Let the *value* of the flow be $f(s, V) = f(V, t)$. (It should be easy to see that this equality holds.)

We are interested in edge cuts $[S, \overline{S}]$ such that $s \in S$ and $t \in \overline{S}$, since those are cuts that stop all possible flow from the source to the sink. We let the capacity of a cut be defined as

$$c([S, \overline{S}]) = \sum_{vw \in [S, \overline{S}]} c(vw),$$

and $f([S, \overline{S}])$ is defined analogously.

Theorem 1 (Ford-Fulkerson)**.** *The maximum possible value of a flow is equal to the minimum capacity of a cut.*

Proof. It is easy to see that for any cut $[S, \overline{S}]$, we have

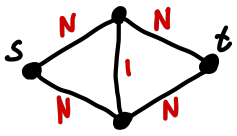
$$f([S, \overline{S}]) \leq c([S, \overline{S}]),$$

so all that remains to be shown is that a flow can always achieve the capacity of *some* cut.

Given a capacity and flow on G, we define an *augmenting path* as a path directed path $e_1, \ldots, e_k$ such that $f(e_i) < c(e_i)$ for $1 \leq i \leq k$. If there is an augmenting path from s to t, we can increase the total flow by adding a constant c to $f(e_i)$, for $1 \leq i \leq k$, as long as $c \leq c(e_i) - f(e_i)$. Suppose that we have performed such augmentations as much as possible, and we are now in a state where there are no augmenting s, t-paths. Since the capacities and flows take integer values, this process is guaranteed to terminate. Let S be the set of vertices v that there exists an augmenting s, v-path. Clearly, $t \in \overline{S}$. Furthermore, we must have that $f(e) = c(e)$ for each edge across the cut $[S, \overline{S}]$, completing the proof. □

When flows and capacities are not integers (or rationals), we are not guaranteed to terminate the process described above. A small example on 6 vertices is given in [2].

In order to implement the algorithm described above, we see that each step of finding the set S (and iterating if $t \in S$) is a simple version of DFS or BFS, and therefore efficient. However, the number of iterations may not be polynomial in the size of the graph. Consider the following example below. In this example, we may accidentally find augmenting paths $suvt, svut, suvt, svut, \ldots$, increasing the total flow by 1 (or 2) each time. There are other algorithms that avoid this problem, as we are about to see.



Linear programming

Linear programming is a technique for optimizing a linear objective function subject to linear constraints. The typical problem statement can be written in the form:

$$\begin{aligned} \text{find a vector:} \quad & \vec{x} \\ \text{which maximizes:} \quad & \vec{c}^T\vec{x} \\ \text{subject to:} \quad & A\vec{x} \leq \vec{b} \\ & \vec{x} \geq \vec{0}. \end{aligned}$$

If we call the above the *primal* linear program, then we define the *dual* as

$$\begin{aligned} \text{find a vector:} \quad & \vec{y} \\ \text{which minimizes:} \quad & \vec{b}^T\vec{y} \\ \text{subject to:} \quad & A^T\vec{y} \geq \vec{c} \\ & \vec{y} \geq \vec{0}. \end{aligned}$$

If one of the programs is unbounded (does not have a finite solution), then the other is infeasible. The following theorem guarantees that when problems have a bounded solution, they solve the same problem.

Theorem 2 (Strong Duality)**.** *If either of the primal or dual program have feasible and bounded solutions, then*

$$\max\{\vec{c}^T\vec{x} \mid A\vec{x} \leq \vec{b},\ \vec{x} \geq \vec{0}\} = \min\{\vec{b}^T\vec{y} \mid A^T\vec{y} \geq \vec{c},\ \vec{y} \geq \vec{0}\}.$$

If a linear program is in R^n, it can be solved in polynomial time. LP programming has been around for at least as long as the early 1800s, when Fourier introduced a method for solving them, but the problem was not known to be in P until the 70s (proved by Khachiyan). If the solution is restricted to integers, however, LP becomes NP-hard. In fact, *binary integer linear programming* (BILP) is one of Karp's 21 NP-complete problems.

Shortest path problem

It is easy to see that the flow problem can be written as a linear program. Now, we will look at a less obvious example: the shortest path problem. Given a, possibly weighted and/or directed, graph G and two of its vertices s, t, find the length of a shortest path from s to t. We have already seen how to do this efficiently with Dijkstra's algorithm, and now we'll instead do this using linear programming. Again, we consider a flow problem, but instead of finding a maximum value flow, we want to send just one unit of flow from s to t, using as a path of as little cost as possible. First, we may assume that there are no edges of the form vs or tv, for any $v \in V$, as they would never be part of such a shortest path. We let $m = |E|$, $\vec{b} \in \mathbb{R}^{2m}$ be the vector of weights (lengths, if you want to think in terms of distances) on the edges (in either direction), and $\vec{y} \in \mathbb{R}^{2m}$ be the vector of flows on the edges. Order the vertices as $s = v_0, v_1, \ldots, v_n = t$, and order the edges in some order $e_1, e_2, \ldots, e_{2m}$. Then our optimization function is

$$\sum_{i=1}^{2m} b_i y_i = \vec{b}^T \vec{y}.$$

In order to make sure that the optimal solution indeed represents a path from s to t, we also require that

$$\sum_{i:\, e_i = ut \in E, u \in V} y_i \geq 1,$$

to ensure that at least a unit of flow arrives at t, and

$$\sum_{i:\, e_i = uv \in E, u \in V} y_i - \sum_{i:\, e_i = vu \in E, u \in V} y_i \geq 0, \quad \forall v \in V \setminus \{s, t\}.$$

to ensure that no flow is created at any vertex other than the source, so that the only flow that arrives at t has originated at s. Now, we translate this to A^T and $\vec{c}$ in to match the definition above. Then,

$$A_{ij} = \begin{cases} 1, & \text{if } e_i = uv_j, \text{ for some } u \in V, \\ -1, & \text{if } e_i = v_j u, \text{ for some } u \in V, \\ 0, & \text{otherwise,} \end{cases}$$

and we let $\vec{c} = (0, \ldots, 0, 1)^T \in \mathbb{R}^{2m}$. Then $A^T \vec{y} \geq \vec{c}$ captures the linear constraints listed, recalling that there are no edges coming out of t.

Now, consider the dual problem. We have defined $A, \vec{b}$ and $\vec{c}$. What constraints do they give us? In A, the columns represent vertices $V \setminus \{s\}$, while the rows represented edges. So the vector $\vec{x}$ represents some nonnegative value on the vertices. The inequality $A\vec{x} \leq \vec{b}$ now defines the constraints

$$x_i - x_j \leq b_k, \;\; 1 \leq k \leq 2m, e_k = v_j v_i.$$

This can be visualized as placing vertex s at position 0 on $\mathbb{R}$ and each v_i at position x_i for $1 \leq i \leq n$, with the constraints ensuring that $x_j - x_i$ does not exceed the weight of the edge $x_i x_j$. Now, maximizing $\vec{c}^T \vec{x}$ simply maximizes x_n, which is the distance between s and t. Clearly, this is bounded by the shortest path between them. Notice that if there is no path from s to t, the primal program in this case is unbounded, and the dual infeasible. When there is a shortest path, the solutions are equal.

Homework

This week's HW is one bigger assignment.

- Choose a graph invariant (a number associated to a graph, such as chromatic number, independence number, etc...) or property (such as being a tree, bipartite, planar, etc...). Ideally, find one that we have not discussed in class and that appeals to you. You can use pages such as the ISGCI, House of Graphs,... to find inspiration, or find it in your own work.
- Find out what is known about the complexity of the associated decision problem, both in general and on specific graph classes.
- Formulate a question that you believe is not yet known.

References

[1] Reinhard Diestel. *Graph theory*. Springer Nature, 2025.

[2] Toshihiko Takahashi. The simplest and smallest network on which the ford-fulkerson maximum flow procedure may fail to terminate. *Journal of Information Processing*, 24(2):390–394, 2016.