Algorithm Design XIV

Linear Programming III

Guoqiang Li
School of Software

SHANGHAI JIAO TONG UNIVERSITY

Max-Flow Min-Cut in LP

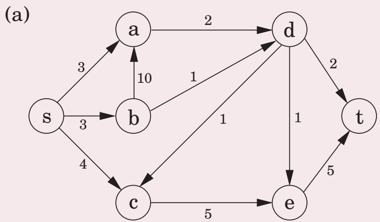
Shipping Oil

We have a network of pipelines along which oil can be sent.

The goal is to ship as much oil as possible from the source to the sink.

Each pipeline has a maximum capacity it can handle, and there are no opportunities for storing oil en route.

A Flow Example



Maximizing Flow

The networks consist of a directed graph $G = (V, E)$; two special nodes $s, t \in V$, a source and sink of G; and capacities $c_e > 0$ on the edges.

Aim to send as much oil as possible from s to t without exceeding the capacities of any of the edges.

Maximizing Flow

A flow consists of a variable f_e for each edge e of the network, satisfying the following two properties:

1. It doesn't violate edge capacities: $0 \leq f_e \leq c_e$ for all $e \in E$.
2. For all nodes u except s and t, the amount of flow entering u equals the amount leaving

$$\sum_{(w,v)\in E} f_{wu} = \sum_{(u,z)\in E} f_{uz}$$

In other words, flow is conserved.

Maximizing Flow

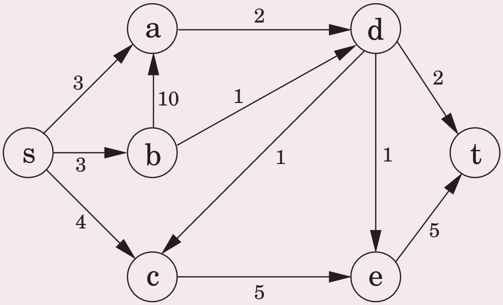
The value of a flow is the total quantity sent from s to t and, by the conservation principle, is equal to the quantity leaving s:

$$\texttt{val}(f) = \sum_{(s,u)\in E} f_{su}$$

Our goal is to assign values to $\{f_e | e \in E\}$ that will satisfy a set of linear constraints and maximize a linear objective function.

This is a linear program. The maximum-flow problem reduces to linear programming.

The Example



11 variables, one per edge.

maximize $f_{sa} + f_{sb} + f_{sc}$

27 constraints:

- 11 for nonnegativity (such as $f_{sa} \geq 0$),
- 11 for capacity (such as $f_{sa} \leq 3$),
- 5 for flow conservation (one for each node of the graph other than s and t, such as $f_{sc} + f_{dc} = f_{ce}$).

Another Representation

First, introduce a fictitious edge of infinite capacity from t to s thus converting the flow to a circulation;

The objective is to maximize the flow on this edge, denoted by f_{ts}.

The advantage of making this modification is that we can now require flow conservation at s and t as well.

Another Representation

$$
\begin{aligned}
& \max f_{ts} && \\
& f_{ij} \le c_{ij} && (i,j) \in E \\
& \sum_{(w,i)\in E} f_{wi} - \sum_{(i,z)\in E} f_{iz} \le 0 && i \in V \\
& f_{ij} \ge 0 && (i,j) \in E
\end{aligned}
$$

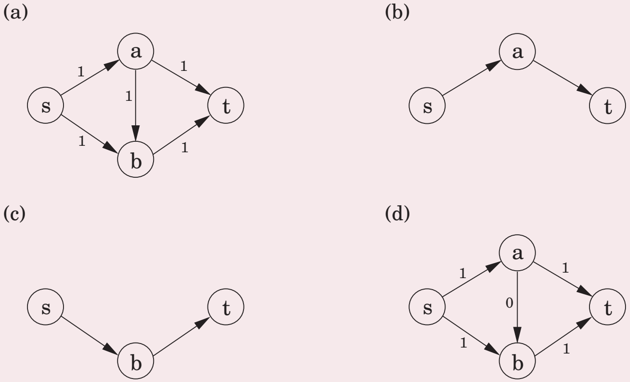
A Closer Look at the Algorithm

Simplex algorithm keeps making local moves on the surface of a convex feasible region, successively improving the objective function until reaches the optimal solution.

The behavior of the simplex has an elementary interpretation:

- Start with zero flow.
- Repeat: choose an appropriate path from s to t, and increase flow along the edges of this path as much as possible.

A Flow Example



A Closer Look at the Algorithm

What if we choose a path that blocks all other paths?

Simplex gets around this problem by also allowing paths to cancel existing flow.

A Closer Look at the Algorithm

To summarize, in each iteration simplex looks for an $s - t$ path whose edges (u, v) can be of two types:

1. (u, v) is in the original network, and is not yet at full capacity.
2. The reverse edge (v, u) is in the original network, and there is some flow along it.

If the current flow is f, then in the first case, edge (u, v) can handle up to $c_{uv} - f_{uv}$ additional units of flow;

in the second case, up to f_{vu} additional units (canceling all or part of the existing flow on (v, u)).

A Closer Look at the Algorithm

These flow-increasing opportunities can be captured in a residual network $G^f = (V, E^f)$, which has exactly the two types of edges listed, with residual capacities c^f:

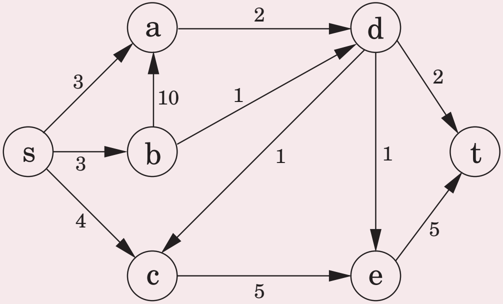
$$\begin{cases} c_{uv} - f_{uv} & \text{if } (u,v) \in E \text{ and } f_{uv} < c_{uv} \\ f_{vu} & \text{if } (v,u) \in E \text{ and } f_{vu} > 0 \end{cases}$$

Thus we can equivalently think of simplex as choosing an $s - t$ path in the residual network.

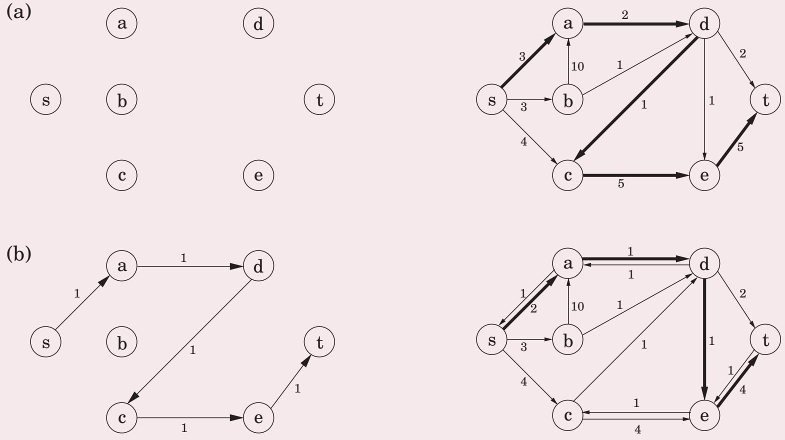
By simulating the behavior of simplex, we get a direct algorithm for solving max-flow.

It proceeds in iterations, each time explicitly constructing G^f, finding a suitable $s - t$ path in G^f by the breadth-first search, and halting if there is no longer any such path along which flow can be increased.

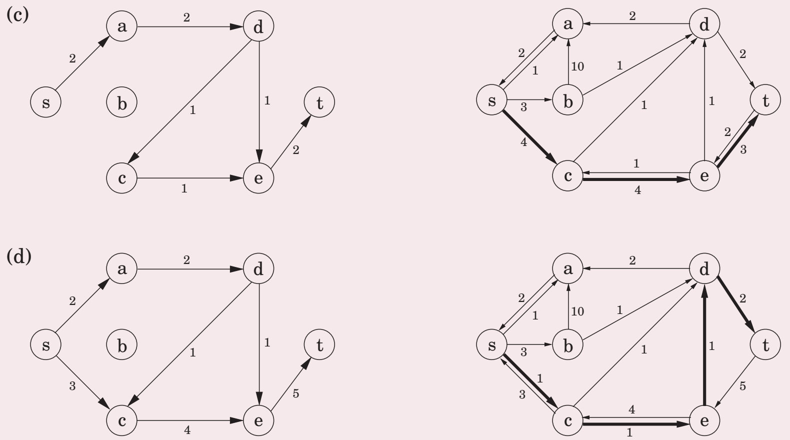
The Example



A Flow Example



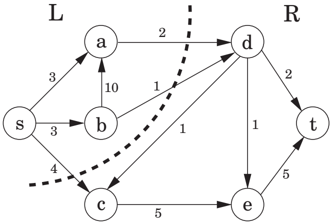
A Flow Example



Cuts

A truly remarkable fact:

Not only does simplex correctly compute a maximum flow, but it also generates a short proof of the optimality of this flow!



An (s,t)-cut partitions the vertices into two disjoint groups L and R, such that $s \in L$ and $t \in R$. Its capacity is the total capacity of the edges from L to R, and as argued previously, is an upper bound on any flow:

Pick any flow f and any (s,t)-cut (L,R). Then $\texttt{size}(f) \leq \texttt{capacity}(L,R)$.

A Certificate of Optimality

Theorem (Max-flow min-cut)

The size of the maximum flow in a network equals the capacity of the smallest (s, t)-cut.

A Certificate of Optimality

Proof:

Suppose f is the final flow when the algorithm terminates.

We know that node t is no longer reachable from s in the residual network G^f.

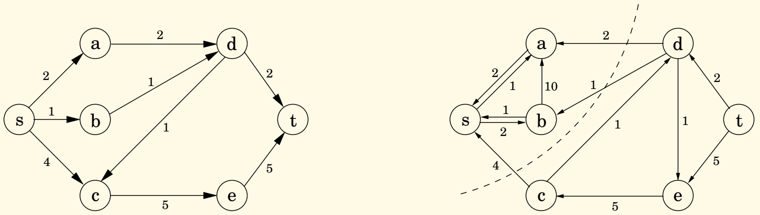
Let L be the nodes that are reachable from s in G^f, and let $R = V \backslash L$ be the rest of the nodes.

We claim that $\texttt{size}(f) = \texttt{capacity}(L, R)$.

To see this, observe that by the way L is defined, any edge going from L to R must be at full capacity (in the current flow f), and any edge from R to L must have zero flow.

Therefore the net flow across (L, R) is exactly the capacity of the cut.

An Example of Max-Flow Min-Cut



Efficiency

Each iteration is efficient, requiring $O(|E|)$ time if a DFS or BFS is used to find an $s - t$ path.

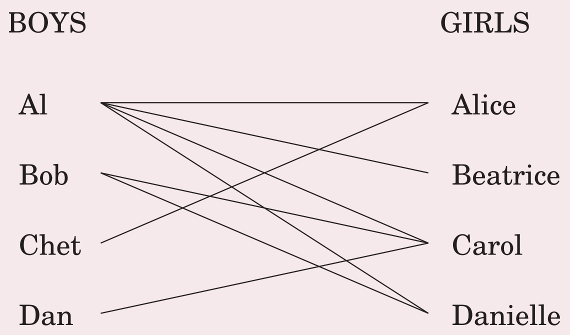
But how many iterations are there?

Suppose all edges in the original network have integer capacities $\leq C$. Then on each iteration of the algorithm, the flow is always an integer and increases by an integer amount. Therefore, since the maximum flow is at most $C|E|$.

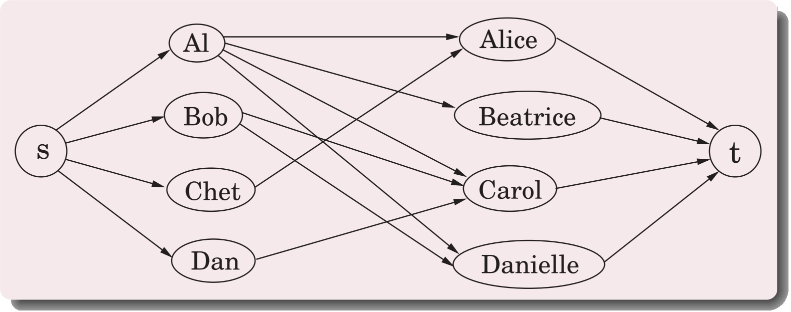
If paths are chosen by using a BFS, which finds the path with the fewest edges, then the number of iterations is at most $O(|V| \cdot |E|)$. *Edmonds-Karp algorithm*

This latter bound gives an overall running time of $O(|V| \cdot |E|^2)$ for maximum flow.

Bipartite Matching



Bipartite Matching



Min-Max Relations in LP

LP for Max Flow

$$
\begin{aligned}
&\max f_{ts} \\
&f_{ij} \le c_{ij} && (i,j) \in E \\
&\sum_{(w,i)\in E} f_{wi} - \sum_{(i,z)\in E} f_{iz} \le 0 && i \in V \\
&f_{ij} \ge 0 && (i,j) \in E
\end{aligned}
$$

LP-Duality

SHANGHAI JIAO TONG UNIVERSITY

$$
\begin{aligned}
& \max f_{ts} & \\
& f_{ij} \le c_{ij} & (i,j) \in E \\
& \sum_{(w,i)\in E} f_{wi} - \sum_{(i,z)\in E} f_{iz} \le 0 & i \in V \\
& f_{ij} \ge 0 & (i,j) \in E
\end{aligned}
$$

$$
\begin{aligned}
& \min \sum_{(i,j)\in E} c_{ij} d_{ij} & \\
& d_{ij} - p_i + p_j \ge 0 & (i,j) \in E \\
& p_s - p_t \ge 1 & \\
& d_{ij} \ge 0 & (i,j) \in E \\
& p_i \ge 0 & i \in V
\end{aligned}
$$

Explanation of the Dual

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j \geq 0 & \quad (i,j) \in E \\ p_s - p_t \geq 1 & \\ d_{ij} \in \{0,1\} & \quad (i,j) \in E \\ p_i \in \{0,1\} & \quad i \in V \end{aligned}$$

To obtain the dual program we introduce variables d_{ij} and p_i corresponding to the two types of inequalities in the primal.

- d_{ij}: distance labels on edges;
- p_i: potentials on nodes.

Integer Program

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j &\geq 0 \quad && (i,j) \in E \\ p_s - p_t &\geq 1 && \\ d_{ij} &\in \{0,1\} && (i,j) \in E \\ p_i &\in \{0,1\} && i \in V \end{aligned}$$

Let $(\mathbf{d}^*, \mathbf{p}^*)$ be an optimal solution to this integer program.

The only way to satisfy the inequality $p_s^* - p_t^* \geq 1$ with a $0/1$ substitution is to set $p_s^* = 1$ and $p_t^* = 0$.

This solution defines an $s-t$ cut $(X, \overline{X})$, where X is the set of potential 1 nodes, and $\overline{X}$ the set of potential 0 nodes.

Integer Program

$$\min \sum_{(i,j)\in E} c_{ij} d_{ij}$$

$$\begin{aligned} d_{ij} - p_i + p_j &\geq 0 \quad && (i,j) \in E \\ p_s - p_t &\geq 1 && \\ d_{ij} &\in \{0,1\} && (i,j) \in E \\ p_i &\in \{0,1\} && i \in V \end{aligned}$$

Consider an edge (i,j) with $i \in X$ and $j \in \overline{X}$, Since $p_i^* = 1$ and $p_j^* = 0$, and thus $d_{ij}^* = 1$.

The distance label for each of the remaining edges can be set to either 0 or 1 without violating the first constraints.

The objective function value is precisely the capacity of the cut $(X, \overline{X})$, and hence $(X, \overline{X})$ must be a minimum $s - t$ cut.

Relaxation of the Integer Program

The integer program is a formulation of the minimum $s - t$ cut problem.

The dual program can be viewed as a relaxation of the integer program where the integrality constraint on the variables is dropped.

This leads to the constraints $1 \geq d_{ij} \geq 0$ for $(i,j) \in E$ and $1 \geq p_i \geq 0$ for $i \in V$.

The upper bound constraints on the variables are redundant; their omission cannot give a better solution.

We will say that this program is the LP relaxation of the integer program.

Relaxation of the Integer Program

The best fractional $s - t$ cut could have lower capacity than the best integral cut. This does not happen here.

Now, it can be proven that each vertex solution is integral, with each coordinate being 0 or 1.

The constraint matrix of this program is totally unimodular, Thus, the dual program always has an integral optimal solution.

More Examples

Set Cover

Set Cover

- Input: A set of elements U, sets $S_1, \ldots, S_m \subseteq U$
- Output: A selection of the S_i whose union is U.
- Cost: Number of sets picked.

Set Cover

Shanghai Jiao Tong University

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{S}} x_S \\ & \sum_{S : e \in S} x_S \geq 1, && e \in U \\ & x_S \geq 0, && S \in \mathcal{S} \end{aligned}$$

$$\begin{aligned} \max \quad & \sum_{e \in U} y_e \\ & \sum_{e : e \in S} y_e \leq 1, && S \in \mathcal{S} \\ & y_e \geq 0, && e \in U \end{aligned}$$

Set Cover

Set Cover

- Input: A set of elements U, sets $S_1, \ldots, S_m \subseteq U$, and a cost function $c : \mathcal{S} \to \mathbb{Q}^+$.
- Output: A selection of the S_i whose union is U.
- Cost: Sum of costs of set picked.

The special case, in which all subsets are of unit cost, will be called the cardinality set cover problem.

Set Cover

$$
\begin{aligned}
\min \quad & \sum_{S \in \mathcal{S}} c(S) x_S \\
& \sum_{S : e \in S} x_S \geq 1, && e \in U \\
& x_S \geq 0, && S \in \mathcal{S}
\end{aligned}
$$

$$
\begin{aligned}
\max \quad & \sum_{e \in U} y_e \\
& \sum_{e : e \in S} y_e \leq c(S), && S \in \mathcal{S} \\
& y_e \geq 0, && e \in U
\end{aligned}
$$

Quiz: Set Multicover

Each element, e, needs to be covered a specified integer number, r_e, of times.

The objective again is to cover all elements up to their coverage requirements at minimum cost.

Each set can be picked at most once.

Integer Program

Let $r_e \in \mathbb{Z}_+$ be the coverage requirement for each element $e \in U$.

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{S}} c(S) x_S \\ & \sum_{S : e \in S} x_S \geq r_e, && e \in U \\ & x_S \in \{0, 1\}, && S \in \mathcal{S} \end{aligned}$$

Linear Program Relaxation

SHANGHAI JIAO TONG UNIVERSITY

In the LP-relaxation, the constraints $x_S \leq 1$ are no longer redundant.

$$
\begin{aligned}
\min \quad & \sum_{S \in \mathcal{S}} c(S) x_S & \\
& \sum_{S : e \in S} x_S \geq r_e, & e \in U \\
& -x_S \geq -1, & S \in \mathcal{S} \\
& x_S \geq 0, & S \in \mathcal{S}
\end{aligned}
$$

Dual Program

SHANGHAI JIAO TONG UNIVERSITY

The additional constraints in the primal lead to new variables, z_S, in the dual.

$$
\begin{aligned}
\max \quad & \sum_{e \in U} r_e y_e - \sum_{S \in \mathcal{S}} z_S & \\
& \Big(\sum_{e : e \in S} y_e\Big) - z_S \leq c(S), & S \in \mathcal{S} \\
& y_e \geq 0, & e \in U \\
& z_S \geq 0, & S \in \mathcal{S}
\end{aligned}
$$