Algorithms (VII)

The Network Flow

Guoqiang Li

School of Software, Shanghai Jiao Tong University

LP and Its Dual

Primal LP

$$\max c^T \mathbf{x}$$
$$A\mathbf{x} \le b$$
$$\mathbf{x} \ge 0$$

Dual LP

$$\min \mathbf{y}^T b$$
$$\mathbf{y}^T A \ge c^T$$
$$\mathbf{y} \ge 0$$

Primal LP:

$$\max \; c_1 x_1 + \cdots + c_n x_n$$
$$a_{i1}x_1 + \cdots + a_{in}x_n \le b_i \quad \text{for } i \in I$$
$$a_{i1}x_1 + \cdots + a_{in}x_n = b_i \quad \text{for } i \in E$$
$$x_j \ge 0 \quad \text{for } j \in N$$

Dual LP:

$$\min \; b_1 y_1 + \cdots + b_m y_m$$
$$a_{1j}y_1 + \cdots + a_{mj}y_m \ge c_j \quad \text{for } j \in N$$
$$a_{1j}y_1 + \cdots + a_{mj}y_m = c_j \quad \text{for } j \notin N$$
$$y_i \ge 0 \quad \text{for } i \in I$$

LP and Its Dual

$$
\begin{aligned}
\max x_1 + 6x_2 \\
x_1 \leq 200 \\
x_2 \leq 300 \\
x_1 + x_2 \leq 400 \\
x_1, x_2, x_3 \geq 0
\end{aligned}
$$

$$
\begin{aligned}
\min 200y_1 + 300y_2 + 400y_3 \\
y_1 + y_3 \geq 1 \\
y_2 + y_3 \geq 6 \\
y_1, y_2, y_3 \geq 0
\end{aligned}
$$

Shortest Path in LP

In the shortest path problem, we are given a weighted, directed graph $G = (V, E)$, with weight function $w : E \to R$ mapping edges to real-valued weights, a source vertex s, and destination vertex t. We wish to compute the weight of a shortest path from s to t.

Shortest Path in LP

$$\min \sum_{e \in E} w_e x_e$$

$$\sum_{e \in \delta(S)} x_e \geq 1 \quad S \in \mathcal{S}$$

$$x_e \geq 0 \qquad e \in E$$

$$\max \sum_{S \in \mathcal{S}} y_S$$

$$\sum_{S \in \mathcal{S}, e \in \delta(S)} y_S \leq w_e \quad e \in E$$

$$y_S \geq 0 \qquad S \in \mathcal{S}$$

Complementary Slackness

Theorem

Assume LP problem (P) has a solution x^* and its dual problem (D) has a solution y^*.

1. If $x_j^* > 0$, then the j-th constraint in (D) is binding.
2. If the j-th constraint in (D) is not binding, then $x_j^* = 0$.
3. If $y_i^* > 0$, then the i-th constraint in (P) is binding.
4. If the i-th constraint in (P) is not binding, then $y_i^* = 0$.

The Simplex Algorithm

On each iteration, simplex has two tasks:

1. Check whether the current vertex is optimal (and if so, halt).
2. Determine where to move next.

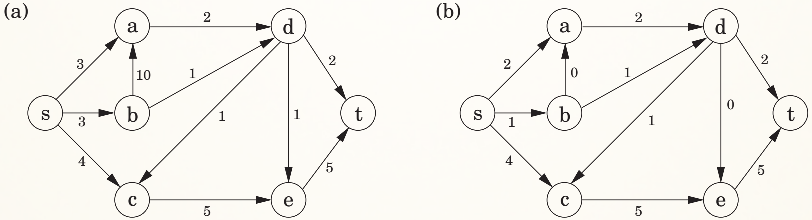
As we will see, both tasks are easy if the vertex happens to be at the origin. And if the vertex is elsewhere, we will transform the coordinate system to move it to the origin!

Flows in Networks

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 s and the sink t.
- 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 $s, t \in V$, which are, respectively, a source and sink of G; and capacities $c_e > 0$ on the edges.
- We would like to send as much oil as possible from s to t without exceeding the capacities of any of the edges.

Maximizing Flow

- A particular shipping scheme is called a flow and 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.

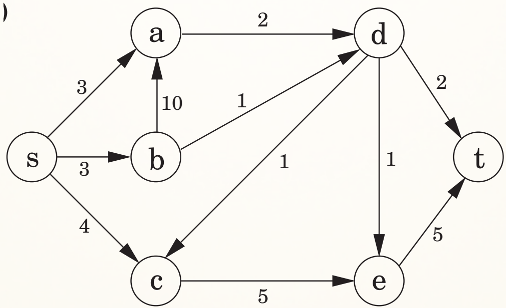
Maximizing Flow

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

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

- In short, 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.
- But this is a linear program! The maximum-flow problem reduces to linear programming.

The Example



LP

- 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 now 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

$$\max f_{ts}$$

$$f_{ij} \leq c_{ij} \qquad (i,j) \in E$$

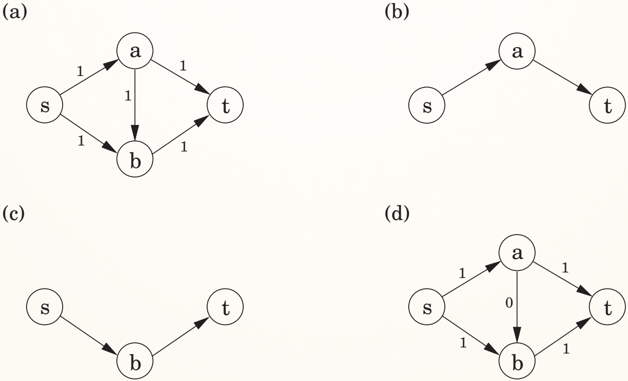
$$\sum_{(w,i)\in E} f_{wi} - \sum_{(i,z)\in E} f_{iz} \leq 0 \quad i \in V$$

$$f_{ij} \geq 0 \qquad (i,j) \in E$$

A Closer Look at the Algorithm

- All we know so far of the simplex algorithm is the vague geometric intuition that it keeps making local moves on the surface of a convex feasible region, successively improving the objective function until it finally reaches the optimal solution.
- The behavior of 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

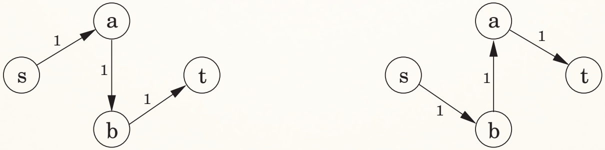


A Closer Look at the Algorithm

- There is just one complication.

  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)).

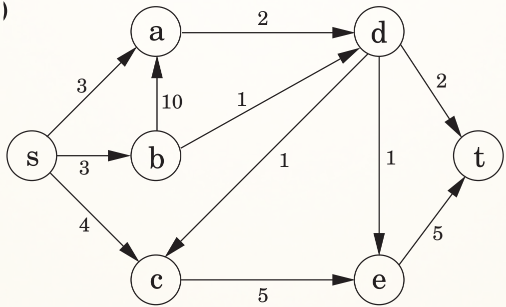
A Closer Look at the Algorithm

- 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:

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

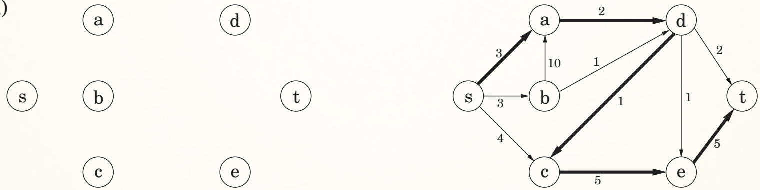
- Thus we can equivalently think of simplex as choosing an $s - t$ path in the residual network.
- By simulating the behavior of simplex, we get a direct algorithm for solving max-flow.
- It proceeds in iterations, each time explicitly constructing G^f, finding a suitable $s - t$ path in G^f by using, say, a linear-time breadth-first search, and halting if there is no longer any such path along which flow can be increased.

The Example

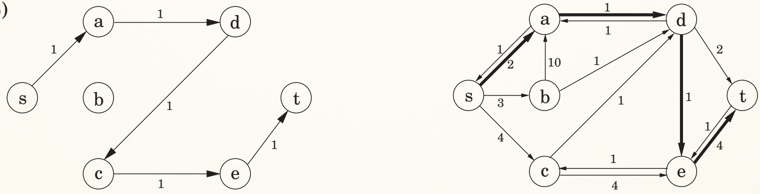


A Flow Example

(a)

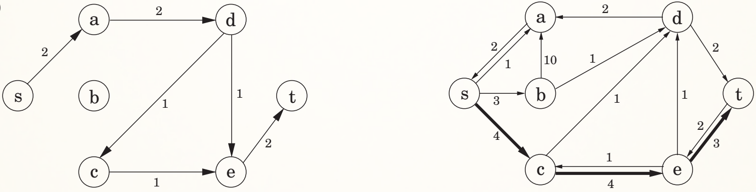


(b)

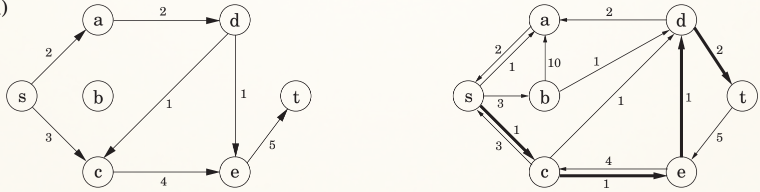


A Flow Example

(c)



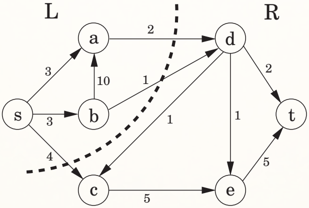
(d)



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 $R = V \setminus L$ 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.

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 $C|E|$, the number of iterations is at most this much.
- If paths are chosen in a sensible manner - in particular, by using a BFS, which finds the path with the fewest edges - then the number of iterations is at most $O(|V| \cdot |E|)$, no matter what the capacities are. *Edmonds-Karp algorithm*
- This latter bound gives an overall running time of $O(|V| \cdot |E|^2)$ for maximum flow.

Efficiency

Lemma:

If the *Edmonds-Karp algorithm* is run on a flow network $G = (V, E)$ with source s and sink t, then for all vertices $v \in V - \{s, t\}$, the shortest path distance $\delta_f(s, v)$ in the residual network G_f increases monotonically with each flow augmentation.

proof:

- Prove by contradiction. Let f be the flow just before the first augmentation that decreases some shortest path distance, and let f' be the flow just afterward.
- Let v be the shortest vertex with the minimum $\delta_{f'}(s, v)$ whose distance was decreased by the augmentation, so that $\delta_{f'}(s, v) < \delta_f(s, v)$. Let $s \rightsquigarrow u \rightarrow v$ be a shortest path from s to v in $G_{f'}$, so that $(u, v) \in E_{f'}$ and

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) - 1$$

Efficiency

proof:

- Because of how we chose v, we know that the distance label of vertex u did not decrease, i.e.,

$$\delta_{f'}(s,u) \geq \delta_f(s,u)$$

- $(u,v) \notin E_f$.
  - otherwise $\delta_f(s,v) \leq \delta_f(s,u) + 1 \leq \delta_{f'}(s,u) + 1 = \delta_{f'}(s,v)$
- $(u,v) \notin E_f$ but $(u,v) \in E_{f'}$? The augmentation must have increased the flow from v to u.
- The Edmonds-Karp algorithm always augments flow along shortest paths, and therefore the shortest path from s to u in G_f has (v,u) as its last edge.
- $\delta_f(s,v) = \delta_f(s,u) - 1 \leq \delta_{f'}(s,u) - 1 = \delta_{f'}(s,v) - 2$.
- Contradict to $\delta_{f'}(s,v) < \delta_f(s,v)$.

Efficiency

Theorem:

If the *Edmonds-Karp algorithm* is run on a flow network $G = (V, E)$ with source s and sink t, then the total number of flow augmentations performed by the algorithm is $O(V \cdot E)$.

Efficiency

proof:

An edge (u, v) in a residual network G_f is critical p if the residual capacity of p is the residual capacity of (u, v), $c_f(p) = c_f(u, v)$. Each of the $|E|$ edges can become critical at most $|V|/2 - 1$ times. Let $(u, v) \in E$,

- Since augmenting paths are shortest paths, when (u, v) is critical for the first time, we have

$$\delta_f(s, v) = \delta_f(s, u) + 1$$

- It reappears after the flow from u to v is decreased, which occurs only if (v, u) appears on an augmenting path. We have

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1$$

Efficiency

proof:

- $\delta_{f'}(s, u) \geq \delta_f(s, u) + 2$.
- The distance of u from the source is initially at least 0. Until u becomes unreachable from the source, if ever, its distance is at most $|V| - 2$. Thus, (u, v) can become critical at most $(|V| - 2)/2 = |V|/2 - 1$ times.
- Since there are $O(E)$ pairs of vertices that can have an edge between them in a residual graph, the total number of critical edges during the entire execution of the Edmonds-Karp algorithm is $O(V \cdot E)$.

Min-Max Relations in LP

LP for Max Flow

$$\max f_{ts}$$

$$f_{ij} \leq c_{ij} \qquad (i,j) \in E$$

$$\sum_{(w,i)\in E} f_{wi} - \sum_{(i,z)\in E} f_{iz} \leq 0 \quad i \in V$$

$$f_{ij} \geq 0 \qquad (i,j) \in E$$

Duality

Primal LP	Dual LP
$\max c^T \mathbf{x}$	$\min \mathbf{y}^T b$
$A\mathbf{x} \leq b$	$\mathbf{y}^T A^T \geq c^T$
$\mathbf{x} \geq 0$	$\mathbf{y} \geq 0$

Primal LP:

$$\begin{gathered}\max\ c_1x_1 + \cdots + c_nx_n \\ a_{i1}x_1 + \cdots + a_{in}x_n \leq b_i \quad \text{for } i \in I \\ a_{i1}x_1 + \cdots + a_{in}x_n = b_i \quad \text{for } i \in E \\ x_j \geq 0 \quad \text{for } j \in N\end{gathered}$$

Dual LP:

$$\begin{gathered}\min\ b_1y_1 + \cdots + b_my_m \\ a_{1j}y_1 + \cdots + a_{mj}y_m \geq c_j \quad \text{for } j \in N \\ a_{1j}y_1 + \cdots + a_{mj}y_m = c_j \quad \text{for } j \notin N \\ y_i \geq 0 \quad \text{for } i \in I\end{gathered}$$

LP-Duality

$$\max f_{ts}$$

$$f_{ij} \leq c_{ij} \qquad (i,j) \in E$$

$$\sum_{(w,i)\in E} f_{wi} - \sum_{(i,z)\in E} f_{iz} \leq 0 \quad i \in V$$

$$f_{ij} \geq 0 \qquad (i,j) \in E$$

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

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

Explanation of the Dual

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

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

- 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

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

- 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 naturally 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

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

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

- 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; however in order to minimize the objective function value it must be set to 0.
- 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.
- Dropping these constraints gives the dual program in the form given above. We will say that this program is the LP relaxation of the integer program.

Relaxation of the Integer Program

- In principle, the best fractional $s - t$ cut could have lower capacity than the best integral cut. Surprisingly enough, this does not happen.
- From linear programming theory we know that for any objective function, i.e., assignment of capacities to the edges of G, there is a vertex solution that is optimal.
- Now, it can be proven that each vertex solution is integral, with each coordinate being 0 or 1.
- This follows from the fact that the constraint matrix of this program is totally unimodular, Thus, the dual program always has an integral optimal solution.

Referred Materials

- Content of this lecture comes from Section 7.2 in [DPV07], Section 26.1 and 26.2 in [CLRS09] and Section 12.2 in [Vaz04].