

Algorithm Design XII

Linear Programming II

Guoqiang Li School of Software



Review of Previous Lecture



$$\max x_1 + 6x_2 + 13x_3$$

$$x_1 \le 200$$

$$x_2 \le 300$$

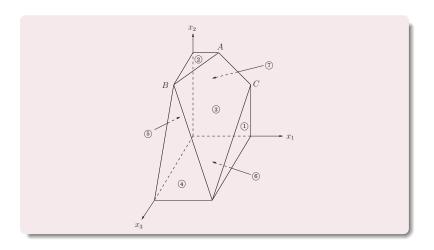
$$x_1 + x_2 + x_3 \le 400$$

$$x_2 + 3x_3 \le 600$$

$$x_1, x_2, x_3 \ge 0$$

The Example







The point of final contact is the optimal vertex: (0, 300, 100), with total profit \$3100.

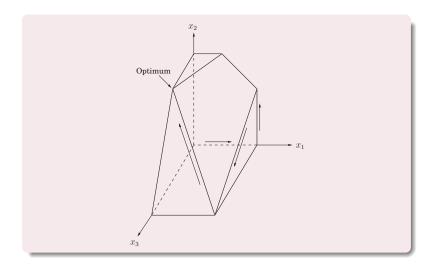
Q: How would the simplex algorithm behave on this modified problem?

A possible trajectory

$$\frac{(0,0,0)}{\$0} \to \frac{(200,0,0)}{\$200} \to \frac{(200,200,0)}{\$1400} \to \frac{(200,0,200)}{\$2800} \to \frac{(0,300,100)}{\$3100}$$

The Example





LP and Its Dual



Primal LP

Dual LP

$$\begin{aligned} \max c^T \mathbf{x} & \min \mathbf{y}^T b \\ A \mathbf{x} &\leq b & \mathbf{y}^T A \geq c^T \\ \mathbf{x} &\geq 0 & \mathbf{y} \geq 0 \end{aligned}$$

Primal LP:

$$\max_{a_{i1}x_{1}+\cdots+a_{in}x_{n}\leq b_{i}} a_{i1}x_{1}+\cdots+a_{in}x_{n}\leq b_{i} \text{ for } i\in I$$

$$a_{i1}x_{1}+\cdots+a_{in}x_{n}=b_{i} \text{ for } i\in E$$

$$x_{j}\geq 0 \text{ for } j\in N$$

$$\min b_1 y_1 + \dots + b_m y_m$$

$$a_{1j} y_1 + \dots + a_{mj} y_m \ge c_j \quad \text{for } j \in N$$

$$a_{1j} y_1 + \dots + a_{mj} y_m = c_j \quad \text{for } j \notin N$$

$$y_i \ge 0 \quad \text{for } i \in I$$

LP and Its Dual



$$\max x_1 + 6x_2 x_1 \le 200 x_2 \le 300 x_1 + x_2 \le 400 x_1, x_2 \ge 0$$

$$\min 200y_1 + 300y_2 + 400y_3$$
$$y_1 + y_3 \ge 1$$
$$y_2 + y_3 \ge 6$$
$$y_1, y_2, y_3 \ge 0$$

Max-Flow Min-Cut in LP



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



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.



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,



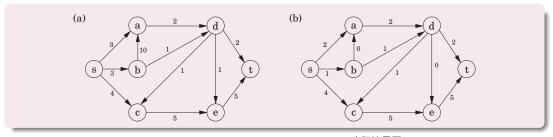
We have a network of pipelines along which oil can be sent.

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







The networks consist of a directed graph G = (V, E);



The networks consist of a directed graph G=(V,E); two special nodes $s,t\in V$, a source and sink of G;



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.



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.



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



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: $0 \le f_e \le c_e$ for all $e \in E$.



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: $0 \le f_e \le c_e$ for all $e \in E$.
- 2 For all nodes u except s and t, the amount of flow entering u equals the amount leaving

$$\sum_{(w,v)\in E} f_{wu} = \sum_{(u,z)\in E} f_{uz}$$
 (即中间结点不会发生流量损失)

In other words, flow is conserved.



The value of a flow is the total quantity sent from \boldsymbol{s} to \boldsymbol{t}



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:

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



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:

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

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.



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:

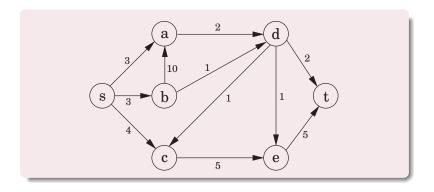
$$\mathtt{val}(f) = \sum_{(\mathbf{s},u) \in E} f_{su}$$

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











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



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

27 constraints:

• 11 for nonnegativity (such as $f_{sa} \ge 0$),



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

27 constraints:

- 11 for nonnegativity (such as $f_{sa} \ge 0$),
- 11 for capacity (such as $f_{sa} \leq 3$),



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

27 constraints:

- 11 for nonnegativity (such as $f_{sa} \ge 0$),
- 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 $f_{sc}+f_{dc}=f_{ce}$).





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



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} .



fictitious: 虚构的

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. 使得之前特殊的节点s和t不再特殊,即与其他节点相同,都要考虑流量的保存。

Another Representation





$$\max f_{ts}$$

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

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

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

添加新的边(t, s)之后,就可以将上述约束条件中的等式全部变为不等式了。因为连接s和t之后成环,只需要满足不等式<=,即可以出现a<=b<=....<=(t, s)<=a的请款,因而最终是会取到等号的,所以两者等价。 通过添加这条新的边,将上述线性规划问题变成一个规范格式的线性规划(max类型),切当上述不等式中等号成立的时候 应该就是一个新的"顶点".



Simplex algorithm keeps making local moves on the surface of a convex feasible region,



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



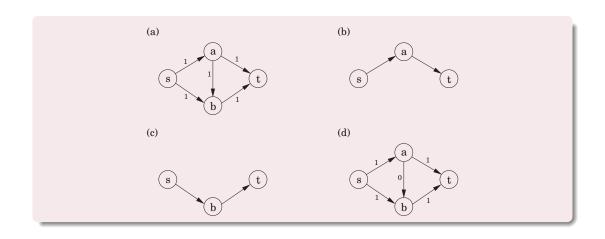
Simplex algorithm keeps making local moves on the surface of a convex feasible region, successively improving the objective function until reaches the optimal solution.

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. (每次选取所选路径中每条边都可能容纳的最大流量)

A Flow Example







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



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

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





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



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

 $\mathbf{0}$ (u, v) is in the original network, and is not yet at full capacity.



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

- $\mathbf{0}$ (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.



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

- $oldsymbol{0}$ (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;



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

- (u, v) is in the original network, and is not yet at full capacity.
- $oldsymbol{0}$ 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)).



These flow-increasing opportunities can be captured in a residual network $G^f = (V, E^f)$,



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 :



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 :

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



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.



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.



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 ,



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,



(剩余网络,余子图)

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 :

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

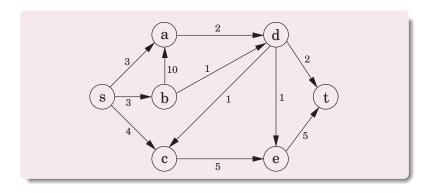
Thus we can equivalently think of simplex as choosing an s-t path in the residual network.

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.

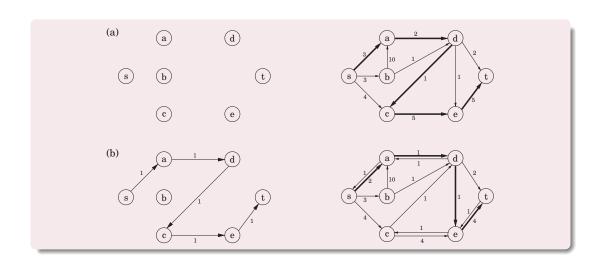
The Example





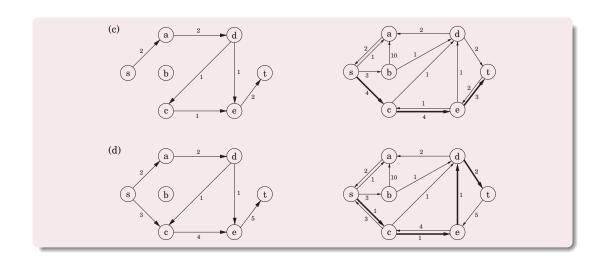
A Flow Example





A Flow Example

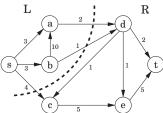






A truly remarkable fact:

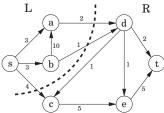
Not only does simplex correctly compute a maximum flow, but it also generates a short proof of the optimality of this flow!





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!

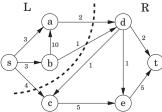


An (s,t)-cut partitions the vertices into two disjoint groups L and R,



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!

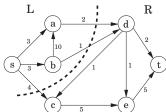


An (s,t)-cut partitions the vertices into two disjoint groups L and R, such that $s \in L$ and $t \in R$. Its capacity is the total capacity of the edges from L to R, and as argued previously, is an upper bound on any flow:



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!



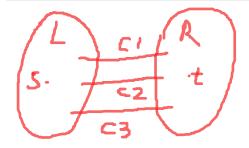
An (s,t)-cut partitions the vertices into two disjoint groups L and R, such that $s \in L$ and $t \in R$. Its capacity is the total capacity of the edges from L to R, and as argued previously, is an upper bound on any flow:

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



Theorem (Max-flow min-cut)

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



在左图中,根据上面PT的描述,可以知道L和R应该是相互之间不可以连接的,但是左图中还存在一条边,是因为这是实际存在的边,而所谓的不可连接指的是在G^f中这几条边已经不联通了,所以说明这几条边的容量已经被全部使用了,所以这各cut的大小为c1+c2+c3,而这就是这个图中流量的上限,因而有flow_size <=c1+c2+c3



Proof:



Proof:

Suppose f is the final flow when the algorithm terminates.



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 .



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.



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 $\operatorname{size}(f) = \operatorname{capacity}(L, R)$.



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 size(f) = 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.



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 size(f) = 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.

根据中间结点没有流量损失可以知道,L中那些在余子图中连接cut反向边的那些节点,在原图中流入的流量只能来源于L中的点,而不能来源于R中的点,因为在原图中只有从L->R的边,所以通过这些连接L和R的边传输的流量就是整张图的总流量。同时也是这个cut的流量大小。

Efficiency



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



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?



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 $\leq C$.



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.



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|.

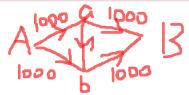


Each iteration is efficient, requiring O(|E|) time if a DFS or BFS is used to find an s-t path.

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





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?

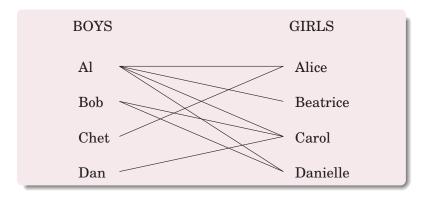
Suppose all edges in the original network have integer capacities $\leq C$. Then on each iteration of the algorithm, the flow is always an integer and increases by an integer amount. Therefore, since the maximum flow is at most C[E]. (但是由于c是可以任意大小的,所以这个算法性能可以是任意坏的)

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

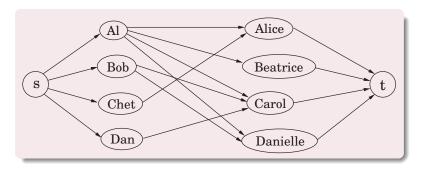




Bipartite Matching



关于如何将一个问题归一化成为一个我们已经学过的问题:



在这问题中,添加一个符合条件的s和t节点之后,计算max flow,并检查结果是否为4;若结果为4,则说明存在最佳匹配,反之则不存在。

Min-Max Relations in LP

LP for Max Flow



$$\max f_{ts}$$

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

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

$$f_{ij} \geq 0$$
 $(i,j) \in E$ U (t, s)

LP-Duality



$$\max f_{ts}$$

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

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

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

LP-Duality



$$\max f_{ts}$$

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

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

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

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

$$d_{ij} - p_i + p_j \ge 0 \quad (i,j) \in E$$

$$p_s - p_t \ge 1 \quad (只对于边(t,s))$$

$$d_{ij} \ge 0 \qquad (i,j) \in E$$

$$p_i \ge 0 \qquad i \in V$$

Explanation of the Dual



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

$$d_{ij} - p_i + p_j \ge 0 \quad (i,j) \in E$$

$$p_s - p_t \ge 1$$

$$d_{ij} \in \{0,1\} \quad (i,j) \in E$$

$$p_i \in \{0,1\} \quad i \in V$$

Explanation of the Dual



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

$$d_{ij} - p_i + p_j \ge 0 \quad (i,j) \in E$$

$$p_s - p_t \ge 1$$

$$d_{ij} \in \{0,1\} \quad (i,j) \in E$$

$$p_i \in \{0,1\} \quad i \in V$$

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;

Explanation of the Dual



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

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.

此处的"potential"指的是**势能**。



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

$$d_{ij} - p_i + p_j \ge 0 \quad (i,j) \in E$$

$$p_s - p_t \ge 1$$

$$d_{ij} \in \{0,1\} \quad (i,j) \in E$$

$$p_i \in \{0,1\} \quad i \in V$$



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

$$d_{ij} - p_i + p_j \ge 0 \quad (i,j) \in E$$

$$p_s - p_t \ge 1$$

$$d_{ij} \in \{0,1\} \quad (i,j) \in E$$

$$p_i \in \{0,1\} \quad i \in V$$

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



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

$$d_{ij} - p_i + p_j \ge 0 \quad (i,j) \in E$$

$$p_s - p_t \ge 1$$

$$d_{ij} \in \{0,1\} \quad (i,j) \in E$$

$$p_i \in \{0,1\} \quad i \in V$$

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

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



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

$$d_{ij} - p_i + p_j \ge 0 \quad (i,j) \in E$$

$$p_s - p_t \ge 1$$

$$d_{ij} \in \{0,1\} \quad (i,j) \in E$$

$$p_i \in \{0,1\} \quad i \in V$$

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

The only way to satisfy the inequality $p_s^* - p_t^* \ge 1$ 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.



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



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

$$d_{ij} - p_i + p_j \ge 0 \quad (i,j) \in E$$

$$p_s - p_t \ge 1$$

$$d_{ij} \in \{0,1\} \quad (i,j) \in E$$

$$p_i \in \{0,1\} \quad i \in V$$

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



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

$$d_{ij} - p_i + p_j \ge 0 \quad (i,j) \in E$$

$$p_s - p_t \ge 1$$

$$d_{ij} \in \{0,1\} \quad (i,j) \in E$$

$$p_i \in \{0,1\} \quad i \in V$$

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.



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

$$d_{ij} - p_i + p_j \ge 0 \quad (i,j) \in E$$

$$p_s - p_t \ge 1$$

$$d_{ij} \in \{0,1\} \quad (i,j) \in E$$

$$p_i \in \{0,1\} \quad i \in V$$

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.



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



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.



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 $1 \ge d_{ij} \ge 0$ for $(i,j) \in E$ and $1 \ge p_i \ge 0$ for $i \in V$.



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 $1 \ge d_{ij} \ge 0$ for $(i,j) \in E$ and $1 \ge p_i \ge 0$ for $i \in V$.

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



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 $1 \ge d_{ij} \ge 0$ for $(i, j) \in E$ and $1 \ge p_i \ge 0$ 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.



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



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 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.

Homework

Homework



Assignment 5(1 week). Exercises 7.6, 7.7, 7.21 and 7.23.