The network simplex algorithm

Combinatorial Optimization

Giovanni Righini

Università degli Studi di Milano

The min cost flow problem

The network simplex algorithm solves LP problems with the combinatorial structure of the min cost flow problem.

$$
\begin{aligned}
\text{minimize } z = & \sum_{(i,j)\in\mathcal{A}} c_{ij}x_{ij} \\
\text{s.t.} \quad & \sum_{j\in\mathcal{N}:(i,j)\in\mathcal{A}} x_{ij} - \sum_{j\in\mathcal{N}:(j,i)\in\mathcal{A}} x_{ji} = b_i \qquad \forall i \in \mathcal{N} \\
& 0 \le x_{ij} \le u_{ij} \qquad \forall (i,j) \in \mathcal{A}.
\end{aligned}
$$

We assume that:

- the flow network $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ is connected;
- all data b, c and u are integer;
- $\sum_{i\in\mathcal{N}} b_i = 0$;
- capacities and costs are non-negative.

Cycle free solutions

Let X be the feasible region.

For each $x \in X$, arcs can be classified as free or restricted:

- $(i,j) \in \mathcal{A}$ is free $\Leftrightarrow 0 < x_{ij} < u_{ij}$,
- $(i,j) \in \mathcal{A}$ is restricted $\Leftrightarrow x_{ij} = 0$ or $x_{ij} = u_{ij}$.

$x \in X$ is cycle-free $\Leftrightarrow x$ contains no cycle made of free arcs.

Along each cycle in a cycle-free solution x we can either increase or decrease the flow, but not both.

Spanning tree solution

A spanning tree solution is defined by a pair (x, T), where

- x is a feasible flow,
- T is a spanning tree on $\mathcal{G}$ (disregarding the orientation of the arcs),

with the property that every arc out of T is restricted (i.e. all free arcs are included in T).

In general, in a spanning tree solution the set of free arcs forms a forest: different spanning trees can include it.
Hence several spanning tree solutions can correspond to the same set of free arcs.

A fundamental property

Property. Apart from pathological cases (unbounded instances), min cost flow instances always have an optimal solution that is a cycle-free and a spanning tree solution.

The network simplex algorithm exploits this property by restricting the search to the spanning tree solutions.

Given a spanning tree solution (x, T) three arc sets are identified:

- T: the arcs in the spanning tree,
- L: non-tree arcs at the lower bound,
- U: non-tree arcs at the upper bound.

If T is made only of free arcs, then (x, T) is non-degenerate.

Relationship between (T, L, U) and the flow

Given a spanning tree structure (T, L, U), one can compute a unique corresponding feasible flow:

- set $x_{ij} = 0 \; \forall (i,j) \in L$
- set $x_{ij} = u_{ij} \; \forall (i,j) \in U$
- solve the system of the flow conservation equations to find $x_{ij} \forall (i,j) \in T$.

If the resulting flow x satisfies the bounds $0 \leq x_{ij} \leq u_{ij} \; \forall (i,j) \in T$, then the spanning tree structure (T, L, U) is feasible.

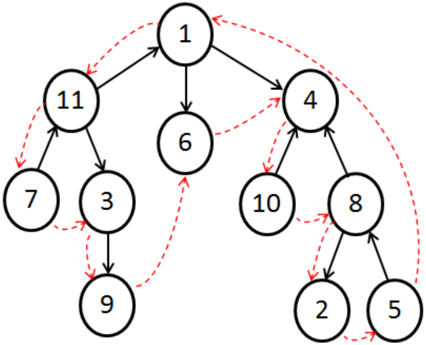
Data structure

The spanning tree T can be represented by three arrays:

- a root node $r \in \mathcal{N}$ is arbitrarily selected;
- $pred(i)$ indicates the predecessor of i in the path from r to i;
- $depth(i)$ indicates the number of arcs between r and i;
- $thread(i)$ points to the successor of i in the sequence generated by a DFS visit of T from r.

For the purpose of computing $pred$, $depth$ and $thread$, the orientation of the arcs in T is neglected.

An example



	1	2	3	4	5	6	7	8	9	10	11
pred	0	8	11	1	8	1	11	4	3	4	1
depth	0	3	2	1	3	1	2	2	3	2	1
thread	11	5	9	10	1	4	3	2	6	8	7

The network simplex algorithm

```
$Initialize(T, L, U)$
$x \leftarrow ComputeFlow(T)$
$y \leftarrow ComputePotentials(T)$
while $OptimalityTest = false$ do
    $(k, l) \leftarrow SelectEnteringArc(T, L, U, x, y)$
    $(p, q) \leftarrow SelectLeavingArc(T, L, U, x, y, (k, l))$
    $Update(T, L, U, x, y, (k, l), (p, q))$
```

Initialization

Option 1.

- Solve a max flow problem from s to t.
- Put all arcs (i,j) with $x_{ij} = 0$ in L.
- Put all arcs (i,j) with $x_{ij} = u_{ij}$ in U.
- Put all the other arcs in T.

Option 2.

- Insert an arc (i,s) with large cost and capacity $\forall i \in \mathcal{N}$ with $b_i > 0$ and set $x_{is} = b_i$.
- Insert an arc (s,i) with large cost and capacity $\forall i \in \mathcal{N}$ with $b_i < 0$ and set $x_{si} = -b_i$.
- Insert all these dummy arcs in T.
- Insert all the original arcs in L.
- Leave U empty.

Computing the flow (1/2)

Part I: flow on arcs in U and L.

```
for $i \in \mathcal{N}$ do
    $b'(i) \leftarrow b(i)$
for $(i,j) \in U$ do
    $x(i,j) \leftarrow u(i,j)$
    $b'(i) \leftarrow b'(i) - u(i,j)$
    $b'(j) \leftarrow b'(j) + u(i,j)$
for $(i,j) \in L$ do
    $x(i,j) \leftarrow 0$
```

Computing the flow (2/2)

Part II: flow on arcs in T. A stack S is used.

```
$j \leftarrow r$; $S \leftarrow \emptyset$
repeat
    $Push(S, j)$; $j \leftarrow thread(j)$
until $j = r$
while $Top(S) \neq r$ do
    $j \leftarrow Pop(S)$
    $i \leftarrow pred(j)$
    if $(i, j) \in T$ then
        $x(i, j) \leftarrow -b'(j)$
    else
        $x(j, i) \leftarrow b'(j)$
    $b'(i) \leftarrow b'(i) + b'(j)$
```

Complexity: $O(m)$.

Primal integrality property. u integer $\Rightarrow$ x integer.

Computing the potentials

Potentials must satisfy: $c_{ij} - y_i + y_j = 0 \;\; \forall (i,j) \in T$.

```
$y(r) \leftarrow 0$
$j \leftarrow thread(r)$
while $j \neq r$ do
    $i \leftarrow pred(j)$
    if $(i,j) \in T$ then
        $y(j) \leftarrow y(i) - c(i,j)$
    else
        $y(j) \leftarrow y(i) + c(i,j)$
    $j \leftarrow thread(j)$
```

Complexity: $O(n)$.

Dual integrality property. c integer $\Rightarrow$ y integer.

Optimality conditions

The reduced cost of each arc $(i,j) \in \mathcal{A}$ is $c^y_{ij} = c_{ij} - y_i + y_j$.

Optimality conditions: (T, L, U) is optimal if and only if $x \in X$ and $\exists y : c^y$ satisfies

- $c^y_{ij} = 0 \ \ \forall (i,j) \in T$;
- $c^y_{ij} \geq 0 \ \ \forall (i,j) \in L$;
- $c^y_{ij} \leq 0 \ \ \forall (i,j) \in U$.

Selection of the entering arc

Eligible arcs are

- arcs $(i,j) \in L$ with $c_{ij}^{y} < 0$;
- arcs $(i,j) \in U$ with $c_{ij}^{y} > 0$.

Any eligible arc can be chosen as an entering arc.

Several policies have been devised for this purpose. For instance:

- Dantzig's rule;
- First eligible arc rule;
- Candidate list rule.

Entering arc selection rules

Dantzig's rule: Select the eligible arc (i, j) that maximizes $|c_{ij}^{y}|$.

Pro: This rule is likely to produce significant improvements per iteration.

Con: It is time-consuming, because it requires to scan all the arcs to find the one with maximum violation.

First eligible arc rule: Scanning the arc list as a circular list, find the first eligible arc.

Pro: With this rule, each iteration is very fast.

Con: It is likely to require many iterations.

Entering arc selection rules

Candidate list rule: Select the eligible arc (i, j) that maximizes $|c^y_{ij}|$ within a restricted candidate list of arcs.

It is a trade-off; it allows for tuning some parameters.

```
// Major iterations //
Scan a circular list of nodes
$\forall i \in N$ insert eligible arcs $(i, j)$ in $\mathcal{L}$
Stop at $\bar{i}$ when $|\mathcal{L}| = \Delta$
Minor iterations
Resume from $\bar{i}$ and loop
```

Parameters: Δ

```
// Minor iterations //
Select $(k, l) \in \mathcal{L}$ with max. violation
Remove $(k, l)$ from the $\mathcal{L}$
Pivot on $(k, l)$
Check all arcs in the $\mathcal{L}$ for eligibility
Stop when $|\mathcal{L}| = 0$ or after $K$ iterations
```

Parameters: K

Selection of the leaving arc

The entering arc (k, l) generates with T a unique cycle W, that can be detected in $O(n)$.

```
$i \leftarrow k$
$j \leftarrow l$
while $i \neq j$ do
    if $depth(i) > depth(j)$ then
        $i \leftarrow pred(i)$
    else
        if $depth(j) > depth(i)$ then
            $j \leftarrow pred(j)$
        else
            $i \leftarrow pred(i)$
            $j \leftarrow pred(j)$
```

Selection of the leaving arc

If $(k,l) \in L$, then W is oriented as (k,l).
If $(k,l) \in U$, then W is oriented opposite to (k,l).

The orientation of W induces a partition of its arcs into

- W^+: forward arcs, where flow is increased;
- W^-: backward arcs, where flow is decreased.

The residual capacities of arcs in W are

$$r_{ij} = \begin{cases} u_{ij} - x_{ij} & \forall (i,j) \in W^+ \\ x_{ij} & \forall (i,j) \in W^-. \end{cases}$$

An amount of flow $\delta = \min_{(i,j)\in W}\{r_{ij}\}$ is sent along W.
This sets the flow of one or more arcs at its lower or upper bound.
These arcs are eligible for being selected as the leaving arc.
Ties can be broken arbitrarily.

If $\delta = 0$, then the iteration is degenerate.

Update

If $\delta = r_{kl} = u_{kl}$, then T does not change. The entering arc is also the leaving arc: it goes from L to U or from U to L.

Otherwise, the leaving arc (p, q) leaves T and enters U or L.

Consider the partition $T = T_1 \cup T_2 \cup (p, q)$, with $r \in T_1$.

For each $i \in T_1$, y_i is unchanged.
For each $i \in T_2$,

- if $(k \in T_1) \wedge (l \in T_2)$, then $y_i \leftarrow y_i - c^y_{kl}$;
- if $(k \in T_2) \wedge (l \in T_1)$, then $y_i \leftarrow y_i + c^y_{kl}$.

Update

```
if $q \in T_2$ then
    $h \leftarrow q$
else
    $h \leftarrow p$
if $k \in T_1$ then
    $\Delta \leftarrow -c^y_{kl}$
else
    $\Delta \leftarrow c^y_{kl}$
$y(h) \leftarrow y(h) + \Delta$
$i \leftarrow thread(h)$
while $depth(i) > depth(h)$ do
    $y(i) \leftarrow y(i) + \Delta$
    $i \leftarrow thread(i)$
```

The arrays pred, depth and thread must also be updated in $O(n)$.

Termination

Degenerate iterations may cause infinite loops. To guarantee finite convergence, further properties are required to the spanning tree T.

Arcs in T, rooted at r, are

- upward arcs, directed to r or
- downward arcs, directed from r.

T is a strongly feasible spanning tree if and only if

- (i, j) empty $\Rightarrow (i, j)$ upward and
- (i, j) saturated $\Rightarrow (i, j)$ downward.

T is a strongly feasible spanning tree if and only if positive flow can be sent from any node to r.

Leaving arc selection rule

W.l.o.g. assume $(k, l) \in L$, i.e. $x_{kl} = 0$.

Then, the pivot cycle W is oriented according to (k, l).

Let w be the apex of W, ie. the first common ancestor of k and l.

Leaving arc selection rule. If more than one bottleneck arc exists in W, select the last bottleneck arc encountered traversing W starting from w.

This rule guarantees that the strong feasiblity property is kept after each pivot step.

Leaving arc selection rule

Let (p, q) be the selected leaving arc.

Partition $W \setminus \{(p, q)\}$ into two parts:

- W_1 from w to (p, q),
- W_2 from (p, q) to w,

according to the orientation of W.

Proof (part I).

No bottleneck arc can exist along W_2, for the selection rule.
Therefore, all nodes in W_2 can send flow to r after the pivot.

Leaving arc selection rule

Proof (part II), case A: $\delta > 0$ along W.

After the pivot, the same amount $\delta > 0$ can be sent back from all nodes in W_1 to r, following the reverse orientation of W_1.

Proof (part II), case B: $\delta = 0$ along W.

All nodes in W_1 before (k, l) can send flow to r after the pivot, because they are unaffected by the pivot.

Nodes in W_1 between (k, l) and (p, q) cannot exist. Before the pivot they could not have been able to send flow to r

- along W_1 because (k, l) was not in T;
- along W_2 because (p, q) is a bottleneck imposing $\delta = 0$.

Therefore all nodes in W_1 can send flow to r after the pivot.

Leaving arc selection rule

Proof (part III). All other nodes $i \notin W$ can send flow to r after the pivot.

If the path from $i \in \mathcal{N}$ to r, $P(i,r)$ does not pass through W, then it is unaffected by the pivot.

If $P(i,r)$ passes through W, let k be the first node of W that is reached along $P(i,r)$.

- The path from i to k is unaffected and can still carry flow;
- $k \in W$ and then it can send flow to r, as proved above.

Finite termination

Keeping the strong feasibility property guarantees finite termination.

Proof.

Non-degenerate iterations are finite, because

- each iteration decreases the total cost by a positive integer amount;
- the total cost is upper bounded by mUC and lower bounded by 0.

Degenerate iterations between two consecutive non-degenerate iterations are finite, because

- each of them decreases the sum of the node potentials by a positive integer amount;
- each node potential is lower bounded by $-nC$.

Dual network simplex algorithm

The dual network simplex algorithm

- keeps flow balance constraints satisfied at each iteration;
- allows for violation of flow bounds on arcs;
- sends flow along cycles in each iteration.

At each iteration the current solution is a triple (T, L, U) such that

- $x_{ij} = 0 \ \forall (i,j) \in L$;
- $x_{ij} = u_{ij} \ \forall (i,j) \in U$;
- but x_{ij} is unrestricted $\forall (i,j) \in T$ (infeasible arcs can exist).

Dual network simplex algorithm

The leaving arc (p, q) is selected first, among the arcs violating the flow bounds.

The violated constraint is repaired by sending a suitable amount of flow along a suitably oriented cycle W.

To produce such a cycle, an entering arc (k, l) is suitably selected.

Dual network simplex algorithm

Consider $T \setminus \{(p,q)\} = T^p \cup T^q$, with $p \in T^p$, $q \in T_q$.

If $x_{pq} > u_{pq}$, flow must be sent from q to p: (k,l) must carry flow from T^p to T^q. Forward and backward arcs are defined as

$$F = \{(i,j) \in A : i \in T^p, j \in T^q\}$$

$$B = \{(i,j) \in A : i \in T^q, j \in T^p\}$$

If $x_{pq} < 0$, flow must be sent from p to q: (k,l) must carry flow from T^q to T^p. Forward and backward arcs are defined as

$$F = \{(i,j) \in A : i \in T^q, j \in T^p\}$$

$$B = \{(i,j) \in A : i \in T^p, j \in T^q\}$$

The set of eligible arcs is

$$Q = (F \cap L) \cup (B \cap U).$$

Dual network simplex algorithm

If $Q = \emptyset$, then the instance is infeasible.

Otherwise,

$$\theta_{ij} = \begin{cases} c^y_{ij} & \forall (i,j) \in F \cap L \\ -c^y_{ij} & \forall (i,j) \in B \cap U \end{cases}$$

and the entering arc is selected as

$$(k,l) = \text{argmin}_{(i,j) \in Q} \{\theta_{ij}\}.$$

A dual pivot is degenerate if and only if $\theta_{kl} = 0$.

Dual network simplex algorithm

If $x_{pq} > u_{pq}$, then $\delta = x_{pq} - u_{pq}$ units of flow are sent along W and (p, q) enters U.
The potential y is decreased by θ_{kl} units for all nodes in T^q.

If $x_{pq} < 0$, then $\delta = -x_{pq}$ units of flow are sent along W and (p, q) enters L.
The potential y is decreased by θ_{kl} units for all nodes in T^p.

The cost of the flow is increased by $\delta\theta_{kl}$ at each iteration.
It is upper bounded by mCU.
Hence the number of non-degenerate iterations is finite.
Degenerate iterations can be avoided by a suitable perturbation of the costs.

Cost perturbation

Arbitrarily sort the arcs and add $\left(\frac{1}{2}\right)^k$ to the cost of each arc in position $k = 1, \ldots, |\mathcal{A}|$.

Then, if x^* is optimal for the perturbed problem, it is also optimal for the original problem.

Proof. If there are no negative cost cycles in the perturbed residual graph, there are no negative cost cycles in the original residual graph, because the cost difference for each cycle is smaller than 1 and the original cost is integer.

In the dual simplex algorithm on the perturbed graph, the reduced cost of each non-tree arc is non-zero.

Proof. The reduced cost a non-tree arc is the cost of the cycle created by inserting the arc in T and no cycle can have an integer perturbed cost.