
15.082 and 6.855J
April 1, 2003

The Global Min Cut problem

Global Min cut

INPUT: A network $G = (N, A)$

OUTPUT: A cut $(S, N \setminus S)$ such that $\text{cap}(S, N \setminus S)$ is minimum.

Note: We do not assume that there is a source node s and destination node t .

Typically, but not always, the network is undirected.

An integer programming formulation

Let $x_{ij} = 1$ if there is an arc (i,j) in the tour T
 $x_{ij} = 0$ otherwise.

- ◆ There are two arcs incident to node i
- ◆ For every cutset $(S, N-S)$, there are two arcs from S to $N-S$.

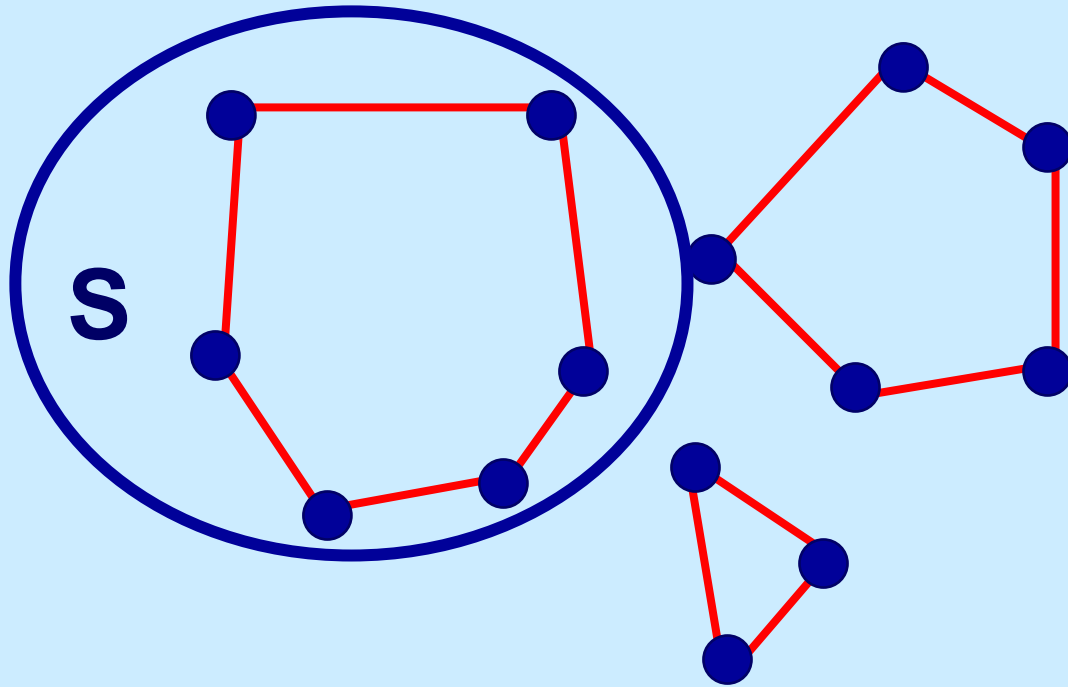
$$\sum_j x_{ij} = 2 \quad \text{for each node } i \quad (1)$$

$$\sum_{i \in S, j \in N \setminus S} x_{ij} \geq 2 \quad \text{for each node set } S \neq N \quad (2)$$

$$x_{ij} = x_{ji} \quad \text{for each } i, j \quad (3)$$

$$0 \leq x_{ij} \leq 1 \quad x_{ij} \text{ integer for each } i, j \quad (4)$$

Any integer solution will be a tour



Any solution to (1) and (3) and (4) will be the union of disjoint directed cycles.

But 2 or more disjoint cycles violates (2).

More on the formulation

Suppose that one has a solution to the linear program satisfying (1), (3) and (4) but relaxing the integrality constraints.

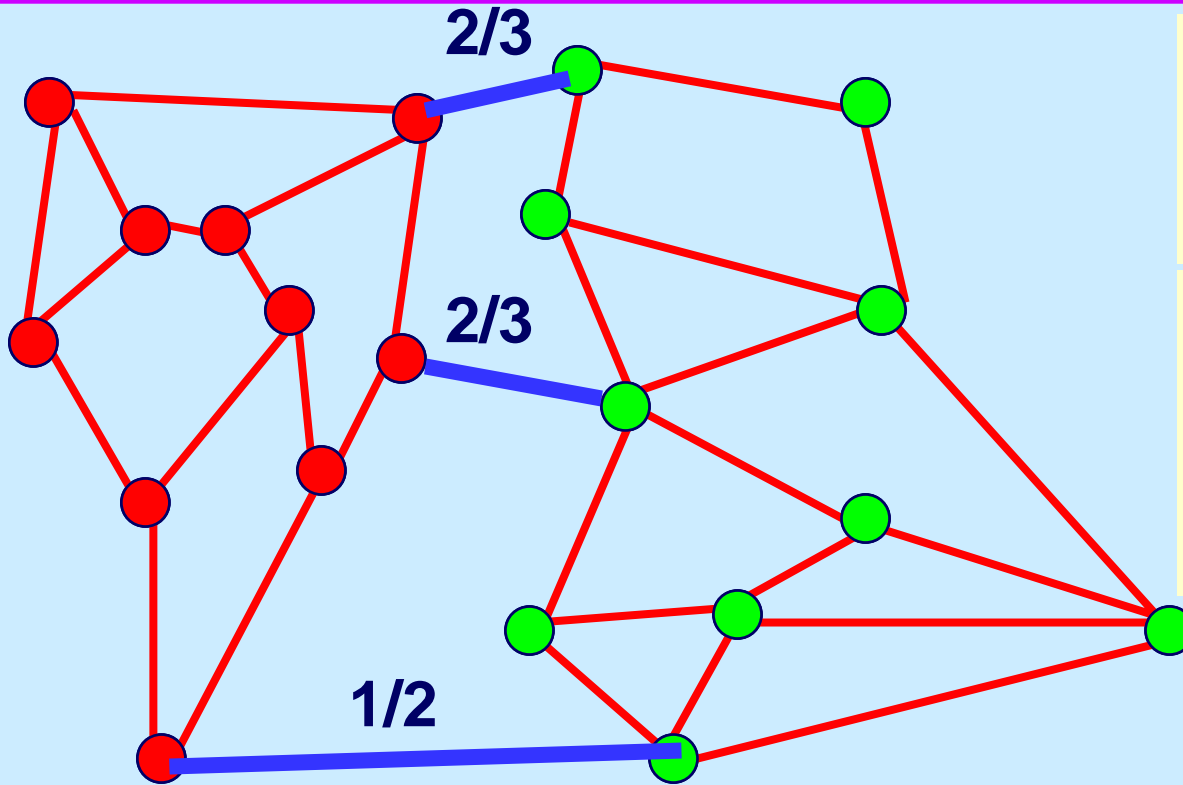
- ◆ **Separation problem:** Either show that all constraints in (2) are satisfied or else determine a violated constraint.

$$\sum_{i \in S, j \in N \setminus S} x_{ij} \geq 2 \quad \text{for each node set } S \neq N$$

Interpretation: each cut has flow at least 2.

Note: the separation problem is repeatedly solved by the best TSP solvers.

The TSP example



Let x be a flow satisfying (1), (3) and (4).

Let $G = (N, A)$ be a graph in which u_{ij} is the value x_{ij} in the LP.

Let (S, T) be the minimum global cut in G .

The solution x is feasible for the LP if and only if the capacity of the cut is at least 2.

In this case, (S, T) is gives a constraint violating (2).

Overview of what the algorithm illustrates

We will solve the problem as a sequence of n min s - t cut problems, with a running time of one s - t cut algorithm.

- ◆ Heuristics can speed up algorithms in practice, and sometimes in theory.**
- ◆ Don't throw away valuable information. Reuse it if possible.**
- ◆ Sometimes subtle changes make really significant differences.**

Algorithm 1. A very simple version

Phase 1: Find the min cut $(S, N \setminus S)$ with $1 \in S$.

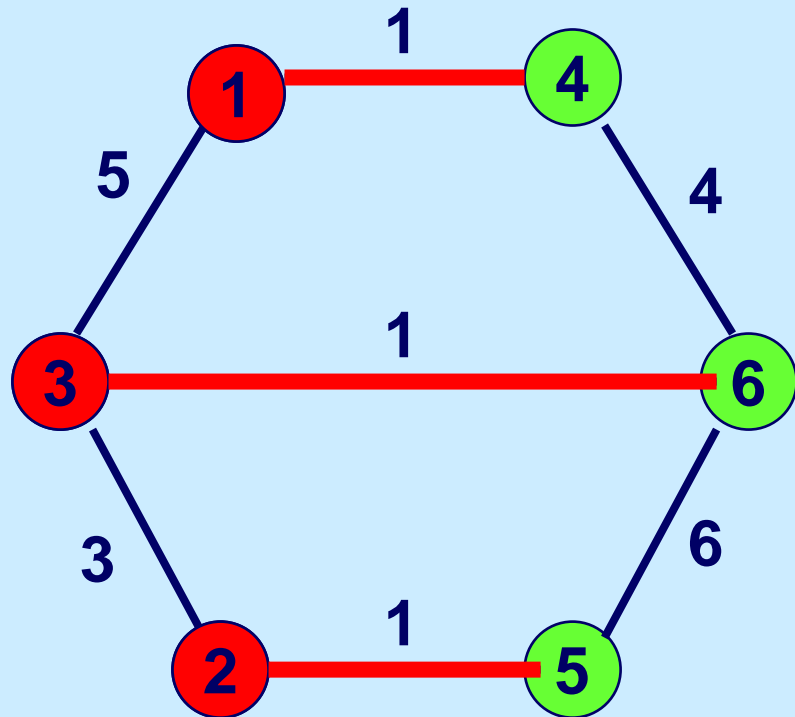
Phase 2: Find the min cut $(S, N \setminus S)$ with $1 \in N \setminus S$

We only describe Phase 1.
Phase 2 is similar.

Algorithm 1.

For $j = 2$ to n find the best 1- j cut.

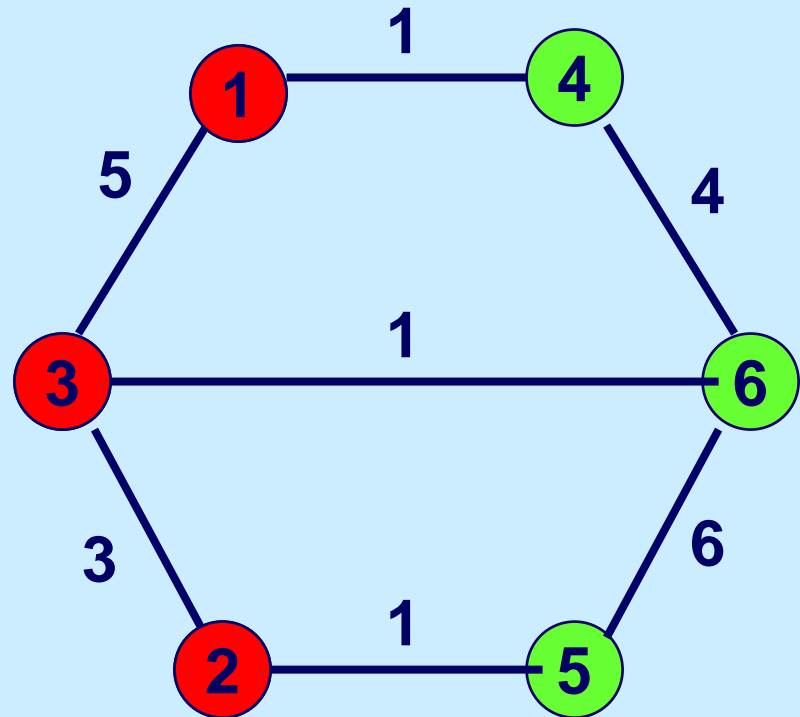
Example: The min cut to the right is the min 1-4 cut (and the min 1-5 cut and the min 1-6 cut).



Algorithm 2

For $j = 2$ to n find the best cut under the restriction that nodes 1 to $j-1$ are all on the S-side of the cut.

Example: The min cut to the right is the min 1-4 cut such that 1, 2, and 3 are all on the same side.

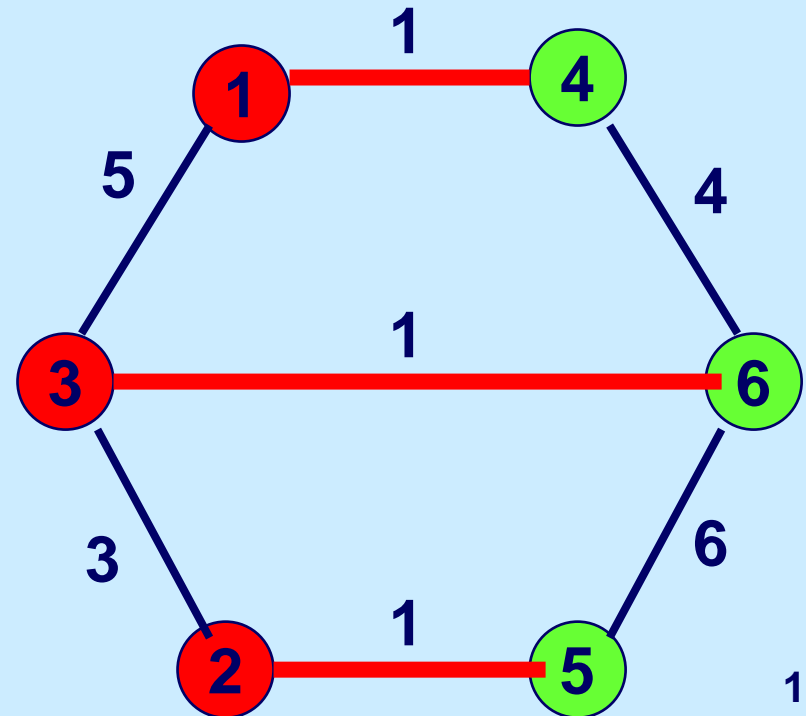


Proof that the algorithm works

First note that each cut $(S, N \setminus S)$ found by the algorithm is a cut in the original network with $1 \in S$.

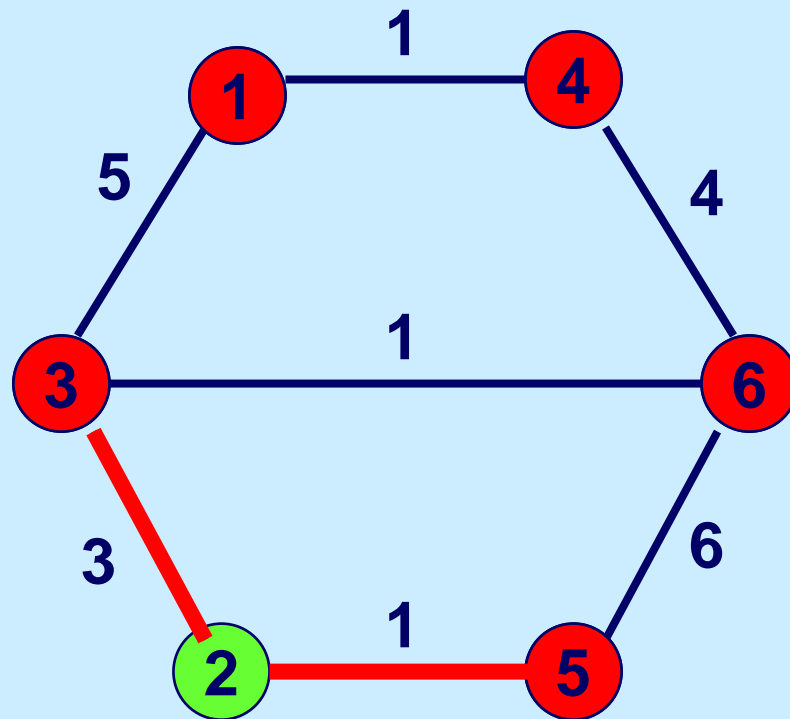
Let (S^*, T^*) be the min cut with $1 \in S^*$. Let j be the least index node that is not on the S^* side of the cut. Then the algorithm finds the min cut in G when it sets j to be the sink.

The min cut in G is the min 1-4 cut subject to 1, 2, 3 all being on the same side of the cut.



Iteration 1. Find the min 1-2 cut

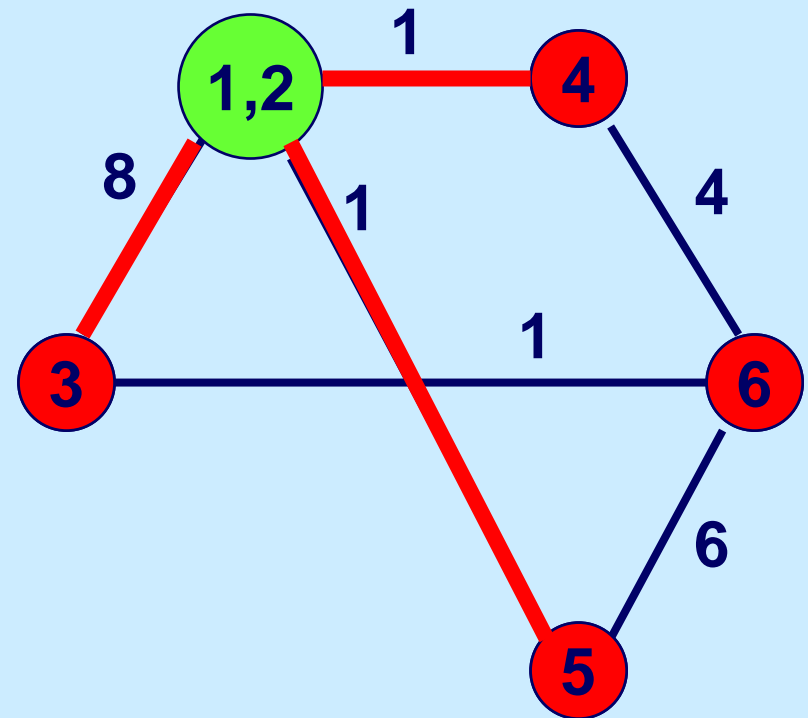
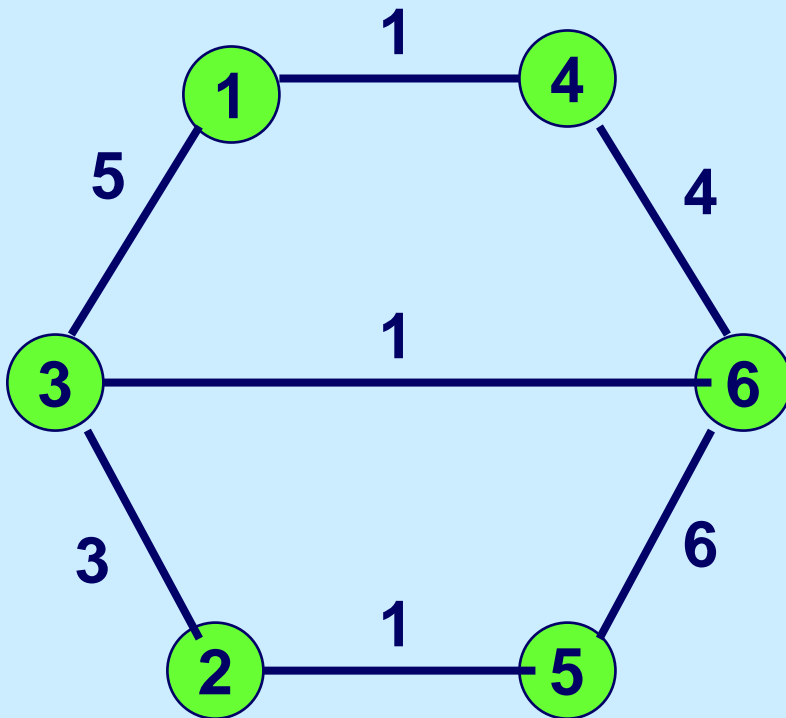
Here is the minimum cut separating node 1 from node 2.



Iteration 2. find the min (1,2)-3 cut

We want to find the min 1-3 cut subject to node 2 is on the same side of the cut as node 1.

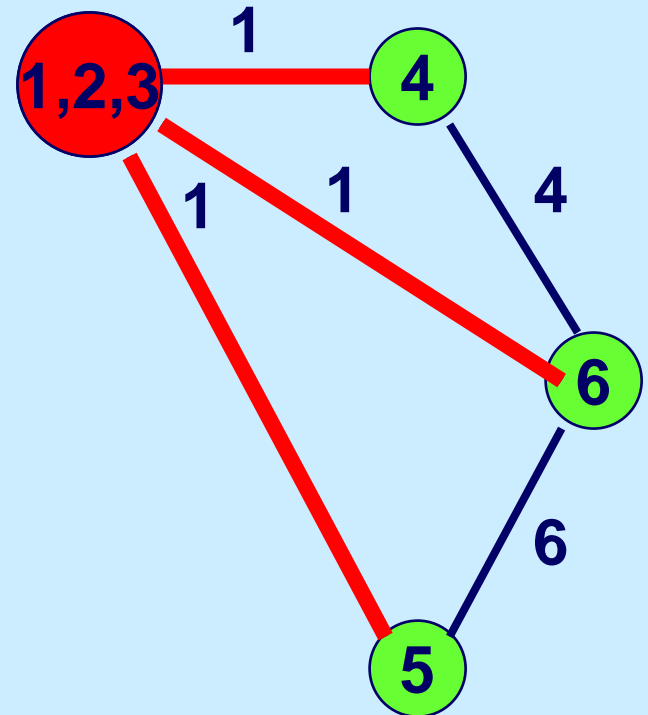
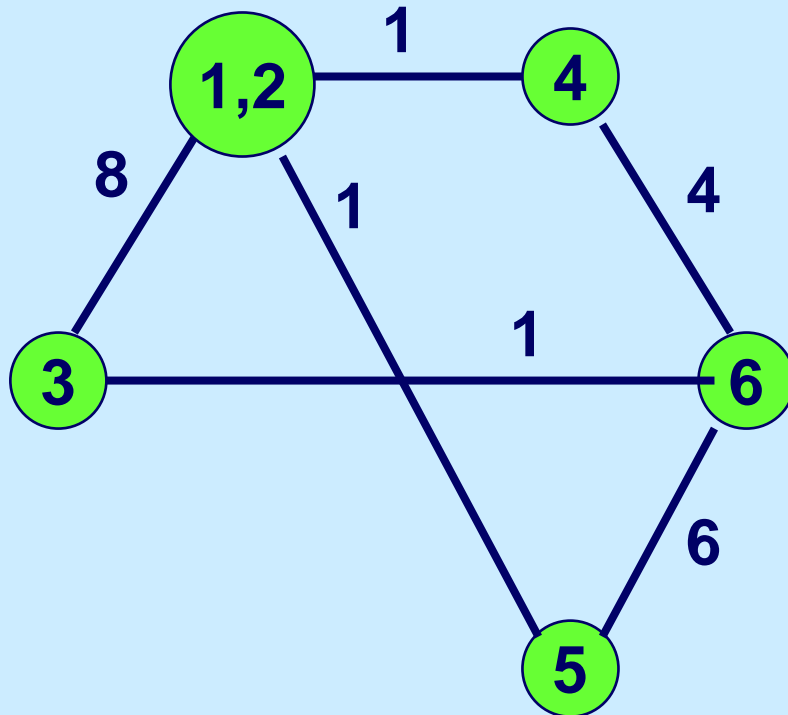
First, merge nodes 1 and 2 into a single node.



Iteration 3. find the min (1,2,3)-4 cut

We want to find the min 1-4 cut subject to nodes 2 and 3 are on the same side of the cut as node 1.

Merge nodes (1,2) and 3 into a single node.

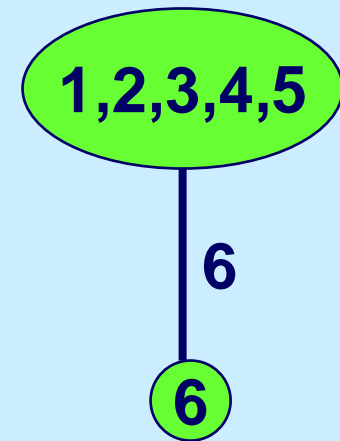
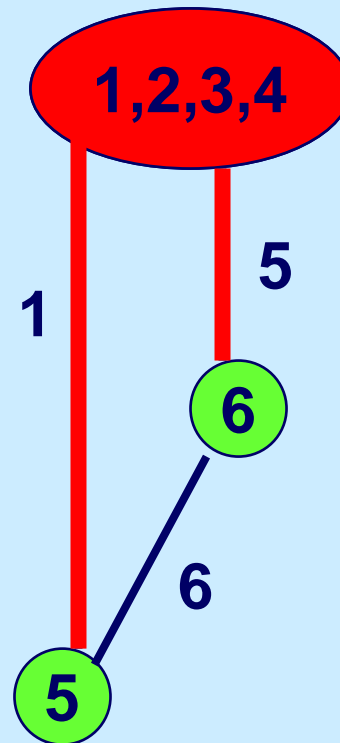
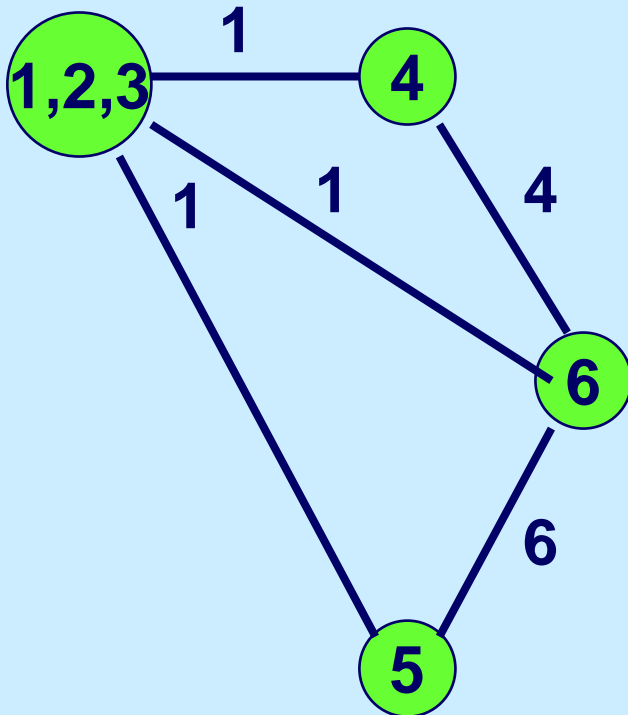


Iterations 4 and 5.

Find the min 1-5 cuts after contracting 1, 2, 3 and 4.

Find the min 1-6 cut after contracting 1, 2, 3, 4, and 5.

Merge nodes (1,2) and 3 into a single node.



Some comments on the algorithm

At first glance, it doesn't look like a speed up.

- ◆ Algorithm 1 solves n “max flow – min cuts”
- ◆ Algorithm 2 solves n “max flow – min cuts”

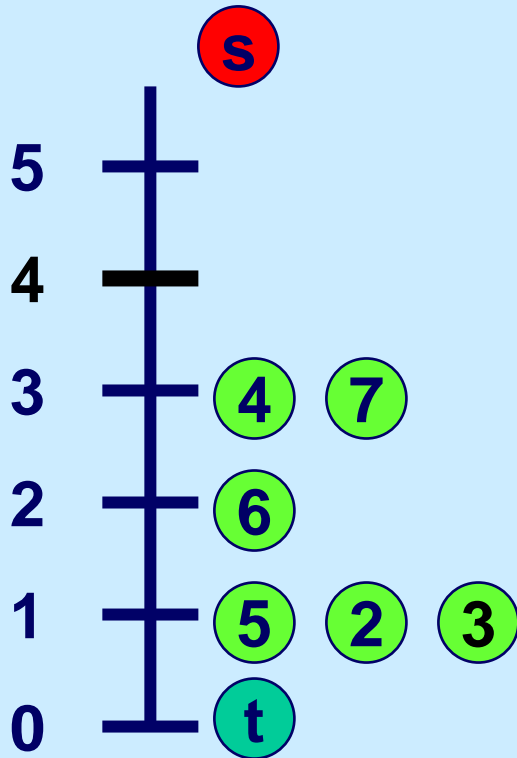
There is some flexibility in which node gets considered as the next sink. (We will choose it carefully.)

It turns out that Algorithm 2 can solve all n min cuts in the time it takes to solve a single max flow.

It relies on preflow push.

Min Global Cut

On distance levels



Distance level k refers to the set of nodes j with $d(j) = k$.

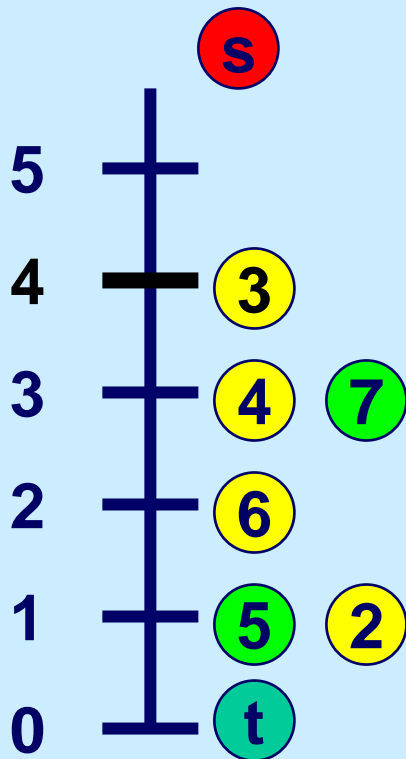
E.g., distance level 3 has nodes 4 and 7.

An **empty distance level** is a level with no nodes.

e.g., levels 4 and 5 are empty

Invariant maintained by the algorithm: the set of non-empty levels k with $k < d(s)$ are consecutive.

Cut Levels and identifying cuts



Active node



Node with no excess

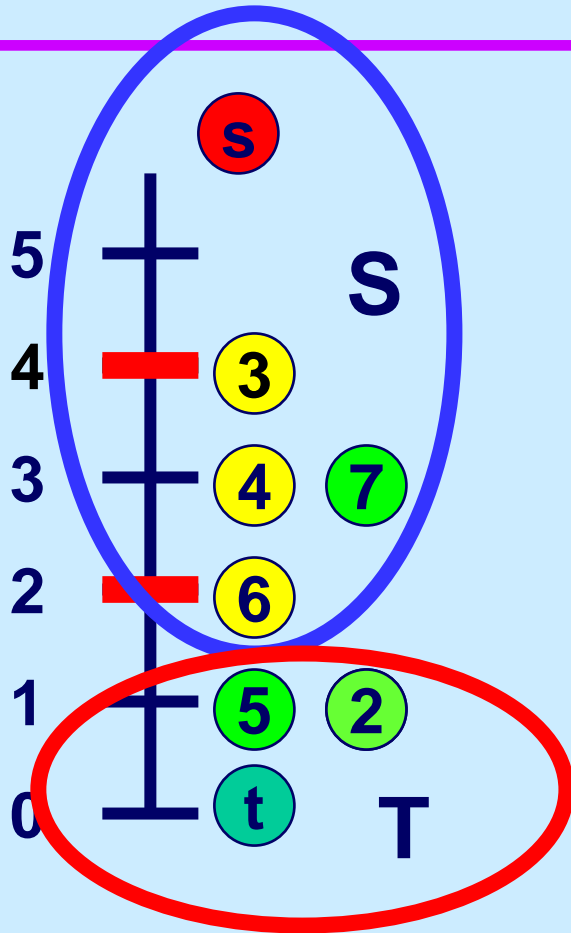
We call a distance level k a cut level if:

1. There is exactly one node j with $d(j) = k$, and
2. There is no admissible arc from node j .

Claim: If distance level k is a cut level, and if $d(i) \geq k$, then there is no path from node i to the sink.

Proof: Let $S = \{ j : d(j) \geq k \}$. Then there are no arcs from S to $N \setminus S$ in $G(x)$, and no path from i to the sink.

Selection rule



Always select an active node j such that $d(j)$ is below the lowest cut level.

Example: suppose that levels 2 and levels 4 are cut levels.

Then one would select node 2.

If node 2 were not active, then the cut (S, T) induced by level 2 would be the min cut.

There is no residual capacity from S to T , and the capacity of (S, T) is the flow into t .

The global min cut algorithm

begin

let $S = \{1\}$, $t = \{2\}$

INITIALIZE

while $S \neq N$ do

begin

find the min S - t cut using preflow push while using the selection rule on the previous slide, and using the relabel rule which prevents empty levels from being created;

let t^* denote some node other than t at distance $\leq d(t) + 1$;

$S := S \cup \{t\}$; saturate arcs out of t ; $d(t) := n$;

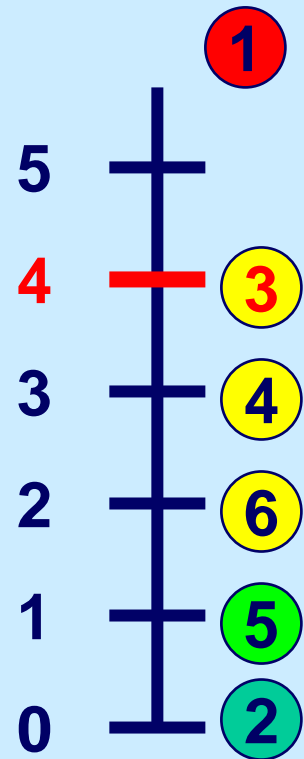
$t := t^*$;

end

end

Min Global Cut

A lemma for the min global cut algorithm



Lemma. *If $j \notin S$, $d(j) \leq n-2$.*

Proof. By induction on the # of nodes in S .

Let k be the node not in S with highest distance level, e.g., $k = 4$.

Then the levels $d(t)$, ..., k are all non-empty. When $S = \{1\}$, $d(t) = 0$, and thus $d(j) \leq k \leq d(t) + (n - |S| - 1) = n-2$.

At each subsequent problem, a node is transferred to S . Thus $|S|$ increases by 1, while $d(t)$ increases by at most 1. Thus $d(j) \leq k < d(t) + (n - |S| - 1) \leq n-2$.

Corollaries

Corollary 1. Each node has its distance label increase fewer than n times.

Corollary 2. Each arc is saturated fewer than n times.

- The same proof works as in preflow push.

Corollary 3. The number of non-saturating pushes is $O(n^2m)$ over all iterations.

- The potential function argument for the preflow push algorithm can be adjusted to work here.

Corollary 4. The dynamic trees implementation solves the global min cut problem in $O(nm \log n)$ time.

Summary

Min Global cut algorithm

- ◆ **n min cuts in the time it takes to find a single max flow**
- ◆ **needs an implementation that does not create empty distance levels**
- ◆ **relies on distance labels being $O(n)$**
- ◆ **application to TSP computations and more**