# C&O 355
# Lecture 23

N. Harvey

# Topics

- Weight-Splitting Method
- Shortest Paths
- Primal-Dual Interpretation
- Local-Ratio Method
- Max Cut

# Weight-Splitting Method

- Let $C \subset \mathbb{R}^n$ be set of feasible solutions to some optimization problem.
- Let $w \in \mathbb{R}^n$ be a "weight vector".
- x is "optimal under w" if x optimizes min $\{ w^T y : y \in C \}$

- **Lemma:** Suppose $w = w_1 + w_2$. Suppose that x is optimal under $w_1$, and x is optimal under $w_2$. Then x is optimal under w.

Frank '81

Hassin '82

# Weight-Splitting Method

- Appears in this paper:

[András Frank]

JOURNAL OF ALGORITHMS 2, 328–336 (1981)

## A Weighted Matroid Intersection Algorithm

ANDRÁS FRANK*

*Research Institute for Telecommunication, Budapest, Hungary*

Received November 5, 1980; revised May 10, 1981

Two matroids $M_1 = (S, \mathcal{I}_1)$ and $M_2 = (S, \mathcal{I}_2)$, and a weight function $\mathbf{s}$ on $S$ (possibly negative or nonintegral) are given. For every nonnegative integer $k$, find a $k$-element common independent set of maximum weight (if it exists).

This problem was solved by J. Edmonds [3, 4] both theoretically and algorithmically. Since then the question has been investigated by a number of different authors; see, for example, [1, 6–10]. The purpose of this note is

# Weight-Splitting Method

- Scroll down a bit…

András Frank

The *weight* of a subset $X$ of $S$ is $\mathbf{s}(X) = \Sigma(\mathbf{s}(x): x \in X)$. If $\mathcal{F}$ is a fami
subsets of $S$ we say that $F \in \mathcal{F}$ is s-maximal in $\mathcal{F}$ if $\mathbf{s}(F) \geq \mathbf{s}(X)$ for $X$

Before describing the algorithm we need some simple lemmas. The
content of the Greedy Algorithm theorem [2] is:

LEMMA 1. *For a given matroid* $M = (S, \mathcal{I})$, *let* $\mathcal{I}^k = \{X: X \in \mathcal{I}, |X| = k\}$. $I \in \mathcal{I}^k$ *is s-maximal in* $\mathcal{I}^k$ *if and only if*

(1) $x \notin I$, $I + x \notin \mathcal{I}$ *imply* $\mathbf{s}(x) \leq \mathbf{s}(y)$, *for every* $y \in C(I, x)$ *and*

(2) $x \notin I$, $I + x \in \mathcal{I}$ *imply* $\mathbf{s}(x) \leq \mathbf{s}(y)$, *for every* $y \in I$,

*This note was written while the author was visiting the University of Waterloo, January–April, 1980.

Weight-Splitting Method was discovered in U. Waterloo C&O Department!

**ShortestPath**( G, S, t, w )

**Input:** Digraph G = (V,A), source vertices $S \subseteq V$, destination vertex $t \in V$, and integer lengths w(a), such that w(a)>0, unless both endpoints of a are in S.

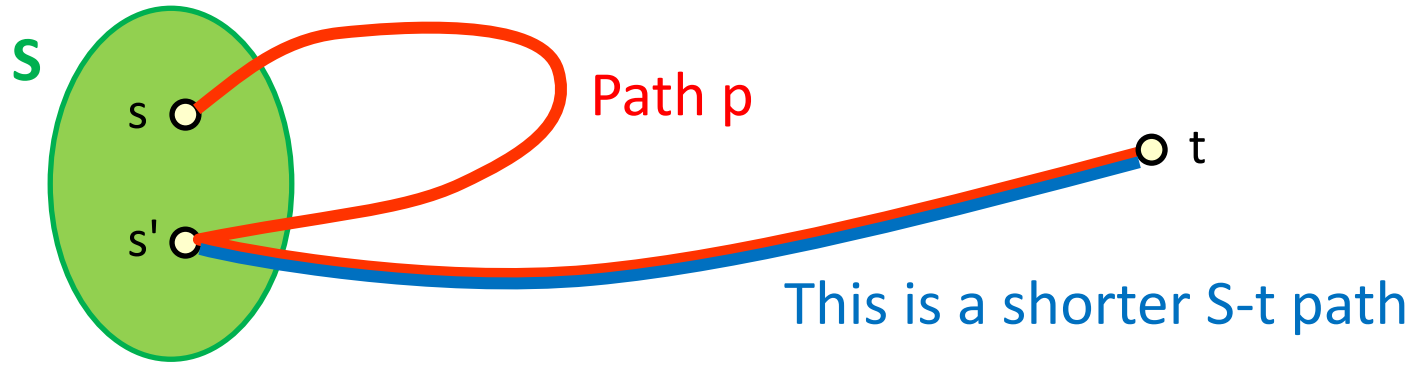
**Output:** A shortest path from some $s \in S$ to t.

- If $t \in S$, return the empty path p=()
- Set $w_1(a)=1$ for all $a \in \delta^+(S)$, and $w^1(a)=0$ otherwise
- Set $w_2 = w - w_1$.
- Set $S' = S \cup \{ u : \exists s \in S$ with $w_2( (s,u) ) = 0 \}$
- Set $p' = (v_1,v_2,\ldots,t) =$ **ShortestPath**( G, S', t, $w_2$ )
- If $v_1 \in S$, then set p=p'
- Else, set $p=(s,v_1,v_2,\ldots,t)$ where $s \in S$ and $w_2( (s,v_1) )=0$
- Return path p

To find shortest s-t path, run ShortestPath(G, {s}, t, w)

# Correctness of Algorithm

- **Claim:** Algorithm returns a shortest path from S to t.
- **Proof:** By induction on number of recursive calls.
- If $t \in S$, then the empty path is trivially shortest.
- Otherwise, p' is a shortest path from S' to t under $w_2$.
- So p is a shortest path from S to t under $w_2$.
  (Note: $\text{length}_{w2}(p)=\text{length}_{w2}(p')$, because if we added an arc, it has $w_2$-length 0.)
- Note: p cannot re-enter S, otherwise a subpath of p would be a shorter S-t path. So p uses exactly one arc of $\delta^+(S)$.



# Correctness of Algorithm

- **Claim:** Algorithm returns a shortest path from S to t.
- **Proof:** By induction on number of recursive calls.
- If $t \in S$, then the empty path is trivially shortest.
- Otherwise, p' is a shortest path from S' to t under $w_2$.
- So p is a shortest path from S to t under $w_2$.
  (Note: $\text{length}_{w_2}(p) = \text{length}_{w_2}(p')$, because if we added an arc, it has $w_2$-length 0.)
- Note: p cannot re-enter S, otherwise a subpath of p would be a shorter S-t path. So p uses exactly one arc of $\delta^+(S)$.
- So $\text{length}_{w_1}(p) = 1$. But **any** S-t path has length **at least** 1 under $w_1$. So p is a shortest path from S to t under $w_1$.
- $\Rightarrow$ p is a shortest S-t path under arc-lengths w, by the Weight-Splitting Lemma. ■

# Another IP & LP for Shortest Paths

- Make variable $x_a$ for each arc $a \in A$
- The IP is:

$$\begin{array}{lll} \min & \sum_{a \in A} w(a) \cdot x_a & \\ \text{s.t.} & \sum_{a \in C} x_a \geq 1 & \forall\ S\text{-}t \text{ cuts } C \\ & x_a \in \{0,1\} & \forall a \in A \end{array}$$

- Corresponding LP & its dual:
  Make variable $y_C$ for each S-t cut C

$$\begin{array}{lll} \min & \sum_{a \in A} w(a) \cdot x_a & \\ \text{s.t.} & \sum_{a \in C} x_a \geq 1 & \forall\ S\text{-}t \text{ cuts } C \\ & x_a \geq 0 & \forall a \in A \end{array} \qquad \begin{array}{lll} \max & \sum_{S\text{-}t \text{ cut } C} y_C & \\ \text{s.t.} & \sum_{C\,:\,a \in C} y_C \leq w(a) & \forall a \in A \\ & y_C \geq 0 & \forall C \end{array}$$

**Theorem:** The Weight-Splitting Algorithm finds optimal primal and dual solutions to these LPs.

**ShortestPath**( G, S, t, w )

**Output:** A shortest path p from S to t, and an optimal solution y for dual LP with weights w

- If $t \in S$
  - Return (p=(), y=0)
- Set $w_1(a)=1$ for all $a \in \delta^+(S)$, and $w_1(a)=0$ otherwise
- Set $w_2 = w - w_1$
- Set $S' = S \cup \{ u : \exists s \in S$ with $w_2( (s,u) ) = 0 \}$
- Set $(p',y') =$ **ShortestPath**$(G,S',t,w_2)$ where $p'=(v_1,v_2,\ldots,t)$
- If $v_1 \in S$
  - Set p=p'
- Else
  - Set $p=(s,v_1,v_2,\ldots,t)$ where $s \in S$ and $w_2( (s,v_1) )=0$
- Set $y_C = 1$ if $C=\delta^+(S)$, otherwise $y_C = y'_C$
- Return (p,y)

# Proof of Theorem

- **Claim:** y is feasible for dual LP with weights w.
- **Proof:**
- By induction, y' feasible for dual LP with weights $w^2$
- So $\sum_{C : a \in C} y'_C \leq w_2(a) \quad \forall a \in A$
- The only difference between y and y' is $y_{\delta^+(S)} = 1$
- So: $\sum_{C : a \in C} y_C = \sum_{C : a \in C} y'_C + [\text{ 1 if } a \in \delta^+(S) ]$

  $\leq w_2(a) + [\text{ 1 if } a \in \delta^+(S) ] = w(a)$
- Clearly y is non-negative
- So y is feasible for dual LP with weights w. □

- Let x be characteristic vector of path p,
  i.e., $x_a$=1 if a$\in$P, otherwise $x_a$=0

- **Note:** x is feasible for primal, since p is an S-t path,
  and its objective value is $w^T x = length_w(p)$

- **Claim:** x is optimal for primal and y is optimal for dual.
- **Proof:** Both x and y are feasible.
- We already argued that:
  $length_{w_2}(p)=length_{w_2}(p')$ and $length_{w_1}(p)=1$

  $\Rightarrow$ $length_w(p) = length_{w_2}(p') + 1$

  $= \sum_C y'_c + 1$

  $= \sum_C y_c$
- So primal objective at x = dual objective at y. □

# How to solve combinatorial IPs?

- Two common approaches
  1. Design combinatorial algorithm that directly solves IP

Often such algorithms have a nice LP interpretation

**Eg:** Weight-splitting algorithm for shortest paths

  2. Relax IP to an LP; prove that they give same solution; solve LP by the ellipsoid method
     - Need to show special structure of the LP's extreme points

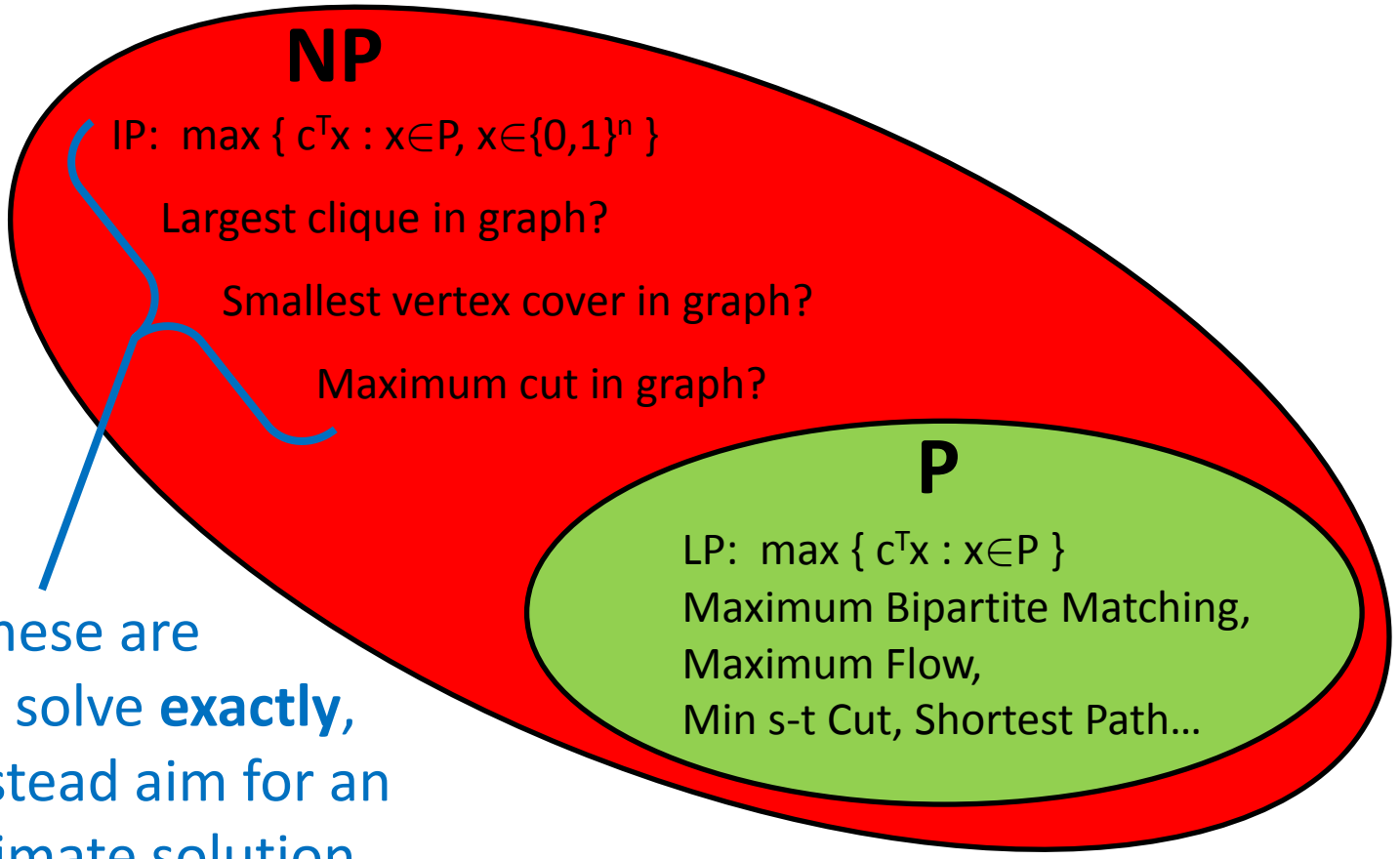
Sometimes we can analyze the extreme points **combinatorially**

     - **Eg:** Perfect matching (in bip. graphs), Min s-t Cut, Max Flow

Sometimes we can use **algebraic** structure of the constraints.

**Eg:** Maximum matching, Vertex cover in bipartite graphs (using TUM matrices)

# Many optimization problems are hard to solve exactly




Since these are hard to solve **exactly**, let's instead aim for an approximate solution

# Approximation Algorithms

- Algorithms for optimization problems that give **provably near-optimal** solutions.
- **Catch-22:** How can you know a solution is *near*-optimal if you don't know the optimum?
- Mathematical Programming to the rescue! Our techniques for analyzing *exact* solutions can often be modified to analyze *approximate* solutions.
  - Eg: Approximate Weight-Splitting
  - Eg: Relax IP to a (non-integral!) LP

# Local-Ratio Method

(Approximate Weight-Splitting)

- Let $C \subset \mathbb{R}^n$ be set of feasible solutions to an optimization problem.
- Let $w \in \mathbb{R}^n$ be a "weight vector".
- x is "r-approximate under w" if $w^T x \geq r \cdot \max \{ w^T y : y \in C \}$
- **Lemma:** Suppose $w = w_1 + w_2$. Suppose that x is r-approximate under both $w_1$ and $w_2$. Then x is r-approximate under w.
- **Proof:**
- Let z be optimal under w. Let $z_i$ be optimal under $w_i$, $i \in \{1,2\}$.
- Then:

$$w^T x = w_1^T x + w_2^T x \geq r \cdot w_1^T z_1 + r \cdot w_2^T z_2$$
$$\geq r \cdot ( w_1^T z + w_2^T z ) = r \cdot w^T z.$$

So x is also r-approximate under w. ■

Bar-Yehuda

Even

# Our Puzzle

- Original Statement:

  There are *n* students in a classroom. Every two students are either enemies or friends. The teacher wants to divide the students into two groups to work on a project while he leaves the classroom. Unfortunately, putting two enemies in the same group will likely to lead to bloodshed. So the teacher would like to partition the students into two groups in a way that maximizes the number of enemies that belong to different groups.

- Restated in graph terminology:

  Let G=(V,E) be a graph with n vertices.

  There is an edge {u,v} if student u and v are enemies.

  For $U \subseteq V$, let $\delta(U) = \{ \{u,v\} : u \in U, v \notin U \}$

  Find a set $U \subseteq V$ such that $|\delta(U)|$ is maximized.

- This is the **Max Cut Problem**:

  $\max\{ |\delta(U)| : U \subseteq V \}$

This is a computationally-hard problem: there is no algorithm to solve it exactly, unless P=NP

# Puzzle Solution

- **Input:** |V| = 750, |E| = 3604 (# enemies)
- **Solutions:**

A Greedy Algorithm

Can cut at most *all* edges

| Who | # Cut Edges | Upper Bound on Optimum | Ratio |
|---|---|---|---|
| Student X | 1897 | 3604 | 52.6% |
| Me | 2962 | 3207 | 92.3% |
| Student Y | 2989 | 3125 | 95.6% |

An algorithm based on Semidefinite Programming (Lecture 24)

Simulated Annealing

*Every* solution cuts at most this many edges

For any odd-cycle of length k, any cut cuts at most k-1 edges.

This bound is based on greedily packing odd-cycles.

# History of Max Cut

- Approximation Algorithms

| Who | Ratio | Technique |
|---|---|---|
| Sahni-Gonzales 1976 | 50% | Greedy algorithm |
| Folklore | 50% | Random Cut |
| Folklore | 50% | Linear Programming |
| Goemans-Williamson 1995 | 87.8% | Semidefinite Programming |
| Trevisan 2009 | 53.1% | Spectral Graph Theory |

- We will see two algorithms:
  - Local-Ratio Method: Also achieves ratio 50%
  - Goemans-Williamson Algorithm (next Lecture)

# Weighted Max Cut

- We can handle the weighted version of the problem
- Let G=(V,E) be complete graph with n vertices.
  For each $e \in E$, there is an integer weight $w(e) \geq 0$

- **Notation:**
  For $U \subseteq V$, let $\delta(U) = \{ \{u,v\} : u \in U, v \notin U \}$
  Let $\delta(U)^T w$ denote $\sum_{e \in \delta(U)} w(e)$

- **Objective:**
  Find a set $U \subseteq V$ such that $\delta(U)^T w$ is maximized

# Sketch of Algorithm

**MaxCut**( G, w )

**Input:** Complete graph G = (V,E), edge weights w

**Output:** $X \subset V$ s.t. $\delta(X)^T w \geq (1/2) \cdot$ optimum

- If $|V|=1$
  - Return $X = \emptyset$
- Else
  - Pick any $v \in V$
  - Let X = **MaxCut**( G\v, w )
  - Return either X or $X \cup \{v\}$, whichever is better

- **Analysis Idea:** Either X or $X \cup \{v\}$ cuts half the weight of edges incident on v. Since this holds for all v, we cut at least half the edges.

# Local-Ratio Algorithm

**MaxCut**( G, w )

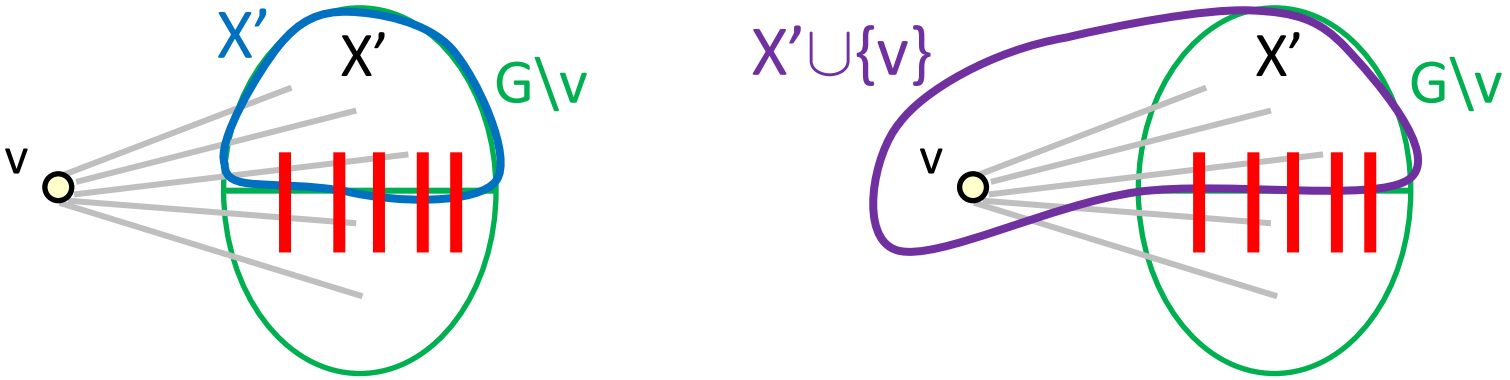
**Input:** Complete graph G = (V,E), edge weights w

**Output:** $X \subset V$ s.t. $\delta(X)^T w \geq (1/2) \cdot$ optimum

- If $|V|=1$
  - Return $X = \emptyset$
- Else
  - Pick any $v \in V$
  - Set $w_1(e)=w(e)$ if e is incident on v, otherwise $w_1(e)=0$
  - Set $w_2 = w - w_1$
  - Let $G' = G \backslash v$
  - Let X' = **MaxCut**( $G \backslash v$, $w_2$ )
  - Return either X' or $X' \cup \{v\}$, whichever is better

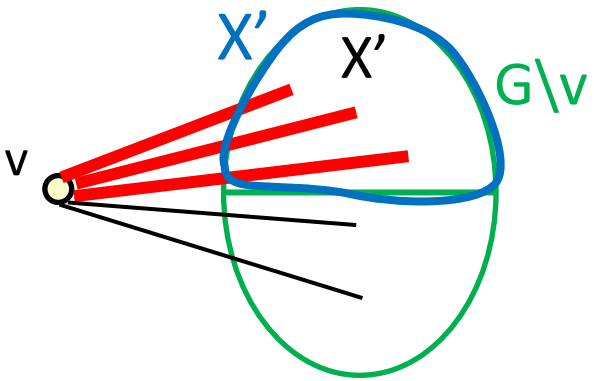
# Correctness of Algorithm

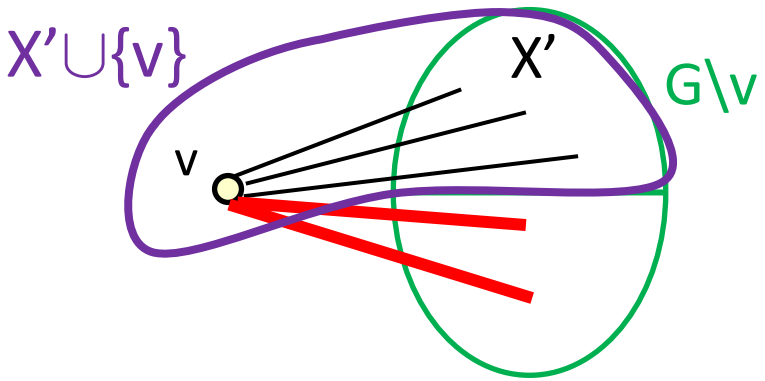
- **Claim:** Algorithm returns $X \subset V$ s.t. $\delta(X)^T w \geq ½$ optimum
- **Proof:** By induction on $|V|$.
- If $|V|=1$, then any cut is optimal.
- By induction, X' is ½-optimal for graph G' with weights $w_2$.
- The edges incident on v have $w_2$-weight zero.
  So **both** X' and $X' \cup \{v\}$ are ½-optimal for G with weights $w_2$.



# Correctness of Algorithm

- **Claim:** Algorithm returns $X \subset V$ s.t. $\delta(X)^T w \geq$ ½ optimum
- **Proof:** By induction on $|V|$.
- If $|V|=1$, then any cut is optimal.
- By induction, X' is ½-optimal for graph G' with weights $w_2$.
- The edges incident on v have $w_2$-weight zero.
  So **both** X' and $X' \cup \{v\}$ are ½-optimal for G with weights $w_2$.
- Any cut U has $w_1$-weight at most $\sum_{e \in E} w_1(e)$
- Every edge incident on v is cut by *either* X' *or* $X' \cup \{v\}$



# Correctness of Algorithm

- **Claim:** Algorithm returns $X \subset V$ s.t. $\delta(X)^T w \geq$ ½ optimum
- **Proof:** By induction on $|V|$.
- If $|V|=1$, then any cut is optimal.
- By induction, X' is ½-optimal for graph G' with weights $w_2$.
- The edges incident on v have $w_2$-weight zero.
  So **both** X' and $X' \cup \{v\}$ are ½-optimal for G with weights $w_2$.
- Any cut U has $w_1$-weight at most $\sum_{e \in E} w_1(e)$
- Every edge incident on v is cut by *either* X' *or* $X' \cup \{v\}$
  So $\delta(X')^T w_1 + \delta(X' \cup \{v\})^T w_1 = \sum_{e \in E} w_1(e) \geq$ optimum under $w_1$
- So *either* X' *or* $X' \cup \{v\}$ is ½-optimal under $w_1$
- So the better of X' or $X' \cup \{v\}$ is ½-optimal under $w_1$ and $w_2$

  $\Rightarrow$ also ½-optimal under w. ■

# What's Next?

- Future C&O classes you could take

| If you liked… | You might like… |
|---|---|
| Max Flows, Min Cuts, Shortest Paths | C&O 351 "Network Flows"<br>C&O 450 "Combinatorial Optimization"<br>C&O 453 "Network Design" |
| Integer Programs | C&O 452 "Integer Programming" |
| Konig's Theorem, Hall's Theorem | C&O 342 "Intro to Graph Theory"<br>C&O 442 "Graph Theory"<br>C&O 444 "Algebraic Graph Theory" |
| Convex Functions,<br>Subgradient Inequality,<br>KKT Theorem | C&O 367 "Nonlinear Optimization"<br>C&O 463 "Convex Optimization"<br>C&O 466 "Continuous Optimization" |
| Semidefinite Programs | C&O 471 "Semidefinite Optimization" |

- If you're unhappy that the ellipsoid method is too slow, you can learn about practical methods in:
  - C&O 466: Continuous Optimization