LP Techniques for Multicuts and Multicommodity Flows (Chs. 18, 20)

André Schumacher

Department of Information and Computer Science
Helsinki University of Technology

March 27, 2008

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Outline

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Primal Problem

$$\begin{aligned} \text{minimise} \quad & \sum_{j=1}^{n} c_j x_j & \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, & i = 1, \ldots, m \\ & x_j \geq 0, & j = 1, \ldots, n \end{aligned}$$

Dual Problem

$$\begin{aligned} \text{maximise} \quad & \sum_{i=1}^{m} b_i y_i & \\ \text{subject to} \quad & \sum_{i=1}^{m} a_{ij} y_i \leq c_j, & j = 1, \ldots, n \\ & y_i \geq 0, & i = 1, \ldots, m \end{aligned}$$

Complementary Slackness Conditions (CS)

Let $\alpha \geq 1$, $\beta \geq 1$.

Primal (Relaxed) CS

$$\forall 1 \leq j \leq n : \quad x_j \neq 0 \quad \longrightarrow \quad \frac{c_j}{\alpha} \leq \sum_{i=1}^{m} a_{ij} y_i \leq c_j$$

Dual (Relaxed) CS

$$\forall 1 \leq i \leq m : \quad y_i \neq 0 \quad \longrightarrow \quad b_i \leq \sum_{j=1}^{n} a_{ij} x_j \leq \beta \cdot b_i$$

Proposition (15.1, page 125)

If x *and* y *are primal and dual feasible satisfying the conditions above then*

$$\sum_{j=1}^{n} c_j x_j \leq \alpha \cdot \beta \cdot \sum_{i=1}^{m} b_i y_i .$$

Recap: Primal-Dual Schema (PDS)

Theorem (12.2, page 96, Weak Duality Theorem)

If x and y are primal and dual feasible solutions, respectively, then

$$\sum_{i=1}^{m} b_i y_i \leq \sum_{j=1}^{n} c_j x_j.$$

Basic idea of PDS:

- Maintain pair of solutions (x, y) that satisfy primal and dual (relaxed) CS, e.g. start with $x = 0$, $y = 0$.
- x may be primal infeasible and y may be dual suboptimal (but dual feasible); primal and dual CS must be satisfied
- Iteratively improve feasibility of x and optimality of y; finally:

$$\sum_{i=1}^{m} b_i y_i \leq \sum_{j=1}^{n} c_j x_j \leq \alpha \cdot \beta \cdot \sum_{i=1}^{m} b_i y_i \leq \alpha \cdot \beta \cdot \text{OPT}.$$

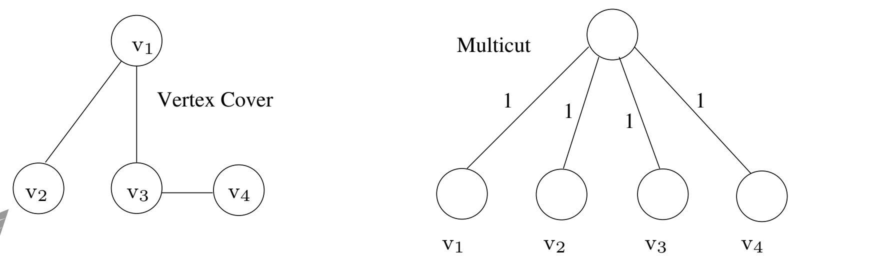
Minimum Multicut Problem (MinIntMulticut)

- Let $G = (V, E)$ be an undirected graph with capacities $c_e \geq 0 \quad \forall e \in E$
- Let $\{(s_1, t_1), \ldots, (s_k, t_k)\}$ be a set of pairs of vertices s.t. $(s_i, t_i) \neq (s_j, t_j) \quad \forall i \neq j$ (called source-sink or source-destination (SD) pairs)
- Multicut M is a set of edges s.t. $M \subseteq E$ and there is no path from s_i to t_i in $(V, E \setminus M) \quad \forall 1 \leq i \leq k$
- Problem: Find minimum capacity multicut in G (generalisation of multiway cut problem)
- First: factor 2 approximation by PDS for trees, then factor $O(\log(k))$ by LP-rounding for general graphs

Minimum Multicut Problem (cont.)

Minimum multicut in trees is NP-hard

- Minimum multicut in trees sounds easy because there is a unique path for each pair (s_i, t_i)
- Problem is NP-hard even for $c_e = 1 \forall e \in E$ and tree height 1
- Idea of reduction of the Minimum Vertex Cover Problem



Edges in graph: $\{\{v_1, v_2\}, \{v_1, v_3\}, \{v_3, v_4\}\}$ SD pairs: $\{(v_1, v_2), (v_1, v_3), (v_3, v_4)\}$

Minimum Multicut Problem (cont.)

- Model the problem with 0/1-integer variables d_e
- For each pair (s_i, t_i), there exists a unique path p_i between s_i and t_i
- Denote by $e \in p_i$ that edge e is on path p_i

ILP Program for MinIntMulticut

$$\begin{aligned} \text{minimise} \quad & \sum_{e \in E} c_e d_e \\ \text{subject to} \quad & \sum_{e \in p_i} d_e \geq 1, & i \in \{1, \ldots, k\} \\ & d_e \in \{0, 1\}, & e \in E \end{aligned}$$

Primal Problem (MinFractMulticut)

$$
\begin{array}{lll}
\text{minimise} & \sum_{e \in E} c_e d_e & \\
\text{subject to} & \sum_{e \in p_i} d_e \geq 1, & i \in \{1, \ldots, k\} \\
 & d_e \geq 0, & e \in E
\end{array}
$$

Dual Problem ≡ Max Multicommodity Flow (MaxFractMulticomFlow)

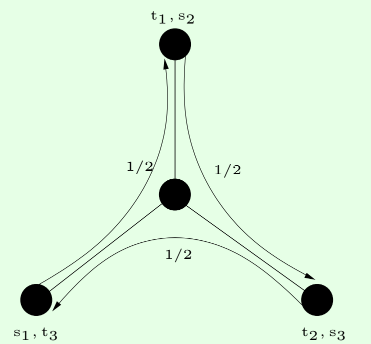
$$
\begin{array}{lll}
\text{maximise} & \sum_{i=1}^{k} f_i & \\
\text{subject to} & \sum_{i: e \in p_i} f_i \leq c_e, & e \in E \\
 & f_i \geq 0, & i \in \{1, \ldots, k\}
\end{array}
$$

Integrality Gap

Example 18.2

Example with unit edge capacities

- MinFractMulticut = MaxFractMulticomFlow = 3/2
- MinIntMulticut = 2
- MaxIntMulticomFlow = 1



Applying the Primal-Dual Schema (PDS)

- Idea for applying PDS: pair (d, f), d primal infeasible(!) “integer multicut”, f dual feasible integral multicommodity flow
- Iteratively improve feasibility of d and optimality of f
- Choosing $\alpha = 1$ and $\beta = 2 \rightarrow$ factor 2 approximation algorithm for MinIntMulticut, factor 1/2 approximation algorithm for MaxIntMulticomFlow

$$\sum_{e \in E}^{n} c_e d_e \leq \alpha \cdot \beta \cdot \sum_{i=1}^{k} f_i \leq \alpha \cdot \beta \cdot \mathrm{OPT}_{\mathrm{MinIntMulticut}}$$

$$\sum_{i=1}^{k} f_i \geq \frac{1}{\alpha \cdot \beta} \sum_{e \in E}^{n} c_e d_e \geq \frac{1}{\alpha \cdot \beta} \cdot \mathrm{OPT}_{\mathrm{MaxIntMulticomFlow}}$$

Applying the Primal-Dual Schema (cont.)

Primal CS

$$\forall e \in E: \quad d_e \neq 0 \implies \sum_{i: e \in p_i} f_i = c_e$$

Any edge picked in the multicut must be saturated.

Relaxed Dual CS

$$\forall i \in \{1, \ldots, k\}: \quad f_i \neq 0 \implies \sum_{e \in p_i} d_e \leq 2$$

At most two edges can be picked from a path carrying nonzero flow. (At least one edge because of primal feasibility at the end.)

Outline of Algorithm

- Root the tree at arbitrary vertex
- Define depth of vertex u to be length of shortest path p to the root (which has depth 0)
- If $e_1, e_2 \in p$, where p is a path from a vertex to the root, and e_1 occurs before e_2, then e_1 is called deeper than e_2
- Denote by $lca(u, v)$ the lowest common ancestor of v and u, i.e. minimum depth vertex on path from u to v
- Start with empty multicut and zero flow
- In each iteration, pick deepest unprocessed vertex v and route greedily integral flow between pairs (s_i, t_i) s.t. $v = lca(s_i, t_i)$

Outline of Algorithm (cont.)

- When no more flow can be routed between these pairs, add all edges saturated in this iteration to list D in arbitrary order, v becomes processed
- Although edge-order within iteration is arbitrary, edges of later iterations are appended to the list
- When all vertices have been processed, the flow is maximal
- As D contains all saturated edges, it is a multicut (but might contain redundant edges)
- Introduce reverse delete step: consider edges in reverse order in which they were added to D, if deletion of edge $e \in D$ still gives valid multicut remove e from D

Algorithm 18.4

1. Initialisation: $f \leftarrow 0; D \leftarrow \emptyset$.
2. Flow routing: For each vertex v, in non-increasing order of depth, do:
   - For each pair (s_i, t_i) s.t. $lca(s_i, t_i) = v$, greedily route integral flow from s_i to t_i.
   - Add to D all edges that were saturated in current iteration in arbitrary order.
3. Let $e_1, e_2, \ldots, e_l$ be the ordered list D
4. Reverse delete: For $j = l$ to $j = 1$ do: If $D \setminus \{e_j\}$ is a multicut in G, then $D \leftarrow D \setminus \{e_j\}$.
5. Output flow and multicut D.

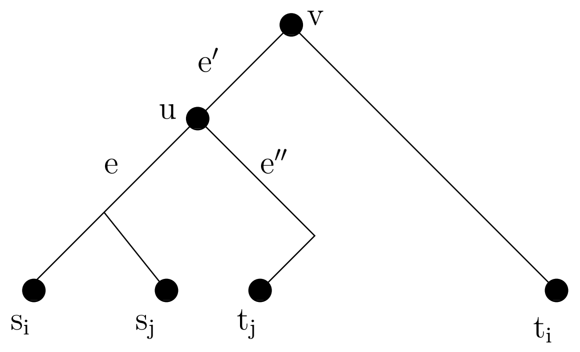
Checking Complementary Slackness

Lemma (18.5, page 149)

Let (s_i, t_i) be a pair with nonzero flow, and let $lca(s_i, t_i) = v$. At most one edge is in D from each of the two paths, s_i to v and t_i to v.

Proof.

Same argument for both paths: Let edges e and e' be picked from path $s_i - v$, while e deeper than e'. Consider moment during reverse delete when edge e is examined. Since e is not discarded, $\exists (s_j, t_j)$, s.t. e is the only edge in D on path $s_j - t_j$. Let $u = lca(s_j, t_j)$. Since e' does not lie on path $s_j - t_j$, it follows u deeper than e' and, hence, v. After u has been processed, D must contain edge e'' from path $s_j - t_j$. (cont.)



Proof. (cont.)

Because nonzero flow was routed on path $s_i - t_i$, e must have been added during the same of later iteration in which v is processed. As v ancestor of u, e is added after e'', therefore, $e'' \in D$ when e is tested. This contradicts the assumption that at this moment e is the only edge in D on path $s_j - t_j$. □

Theorem (18.6, page 150)

Algorithm 18.4 achieves approximation guarantees of factor 2 for MinIntMulticut and 1/2 for MaxIntMulticomFlow on trees.

Proof.

The flow found in step 2 is maximal, and since D contains all saturated edges, D is a multicut. Since the reverse delete only discards redundant edges, D stays a multicut. Thus, multicut and flow solutions are primal and dual feasible, respectively. Since each edge in D is saturated, primal conditions are satisfied. By the previous Lemma, at most two edges have been picked from each path carrying nonzero flow. Therefore dual conditions are also satisfied. □

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Recap: LP-rounding-based Algorithms

- Very simple method
  1. Start with ILP formulation of problem
  2. Relax integer constraints and solve LP
  3. Round up non-integral solution
- Basic idea: rounded solution may not be "too far" from optimal non-integral solution in terms of objective value and thus from the optimal integral solution
- Method was applied to Set Cover Problem in Chapter 14
- Here we apply it to the Multicut Problem in general graphs

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Multicut in General Graphs

- Recall Multicut Problem (in Chapter 20 the dual problem to a primal multicommodity flow problem)
- Let $G = (V, E)$ be an undirected graph with edge capacities $c_e \geq 0$
- Let $\{(s_1, t_1), \ldots, (s_k, t_k)\}$ be a set of pairs of vertices s.t. $(s_i, t_i) \neq (s_j, t_j) \quad \forall i \neq j$ (called source-sink or source-destination (SD) pairs)
- Denote by P_i the set of all paths from s_i to t_i in G and let $P = \bigcup_{i=1}^{k} P_i$.
- Multicut M is a set of edges s.t. $M \subseteq E$ and there is no path from s_i to t_i in $(V, E \setminus M) \quad \forall 1 \leq i \leq k$
- Problem: Find minimum capacity multicut in G (MinIntMulticut)

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Multicut in General Graphs (cont.)

Relaxed Problem (MinFractMulticut)

$$\begin{aligned} \text{minimise} \quad & \sum_{e \in E} c_e d_e & \\ \text{subject to} \quad & \sum_{e \in p} d_e \geq 1, & p \in P \\ & d_e \geq 0, & e \in E \end{aligned}$$

- Generalised version from previous problem: possibly more than one path between each SD pair
- Solving the problem can be interpreted as assigning distance labels (lengths) d_e to edges e, s.t. distance labels satisfy

$$\text{dist}(s_i, t_i) := \min_{p \in P_i} \sum_{e \in p} d_e \geq 1, \quad \forall 1 \leq i \leq k.$$

Outline of Rounding-Based Algorithm

- Obtain approximate solution of MinIntMulticut by rounding optimal solution to MinFractMulticut
- MinFractMulticut can be solved in polynomial time using the ellipsoid algorithm
- Problem provides simple feasibility check: one shortest path computation for each pair
- Let $F = \sum_{e \in E} c_e d_e$, an optimal solution to MinFractMulticut
- Let $D = \{e \in E | d_e > 0\}$; problem: how does one pick edges from D that do not increase the capacity too much? (compared to F)

Outline of Rounding-Based Algorithm (cont.)

- Intuition: edges with large distance labels are more important than those with small labels (because of optimality for MinFractMulticut)
- Basic idea: grow disjoint sets of vertices ("balls", "regions") starting from root nodes such that:
  - regions consist of vertices at distance at most a given value from the root node
  - no region contains both, source and destination, of any pair
  - for each SD pair, either the source or the destination is in one of the regions
  - edges with large distance labels are more likely to lie at the boundary of regions
  - regions are grown one after another
- Edges crossing region boundaries later form the multicut

Additional Notation

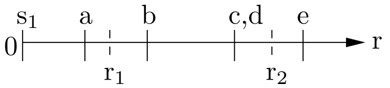
- Define weight of edge e to be $c_e d_e$
- Denote by dist(u, v) the distance of u from v, i.e. the length of the shortest path $u - v$ in G w.r.t. edge lengths d_e
- For $S \subset V$, $\delta(S)$ denotes the set of edges in cut $(S, \bar{S})$, $c(S)$ denote the capacity of the cut
- Consider for now source s_1 to be the root of a region; denote by S(r) the set of vertices at distance at most r, i.e.

$$S(r) = \{v \in V | \text{dist}(s_1, v) \leq r\}, \quad S(0) = \{s_1\}.$$

Continuous Region-Growing Process

Consider varying r continuously and observe changes in $S(r)$ (source s_1 fixed)

$S(r_1) = \{s_1, a\}$
$S(r_2) = \{s_1, a, b, c, d\}$



Lemma (20.2, page 170)

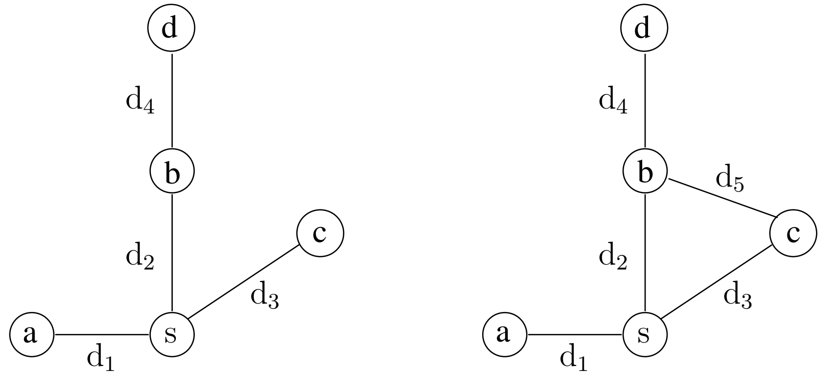
If the region growing process is terminated before radius $r = 1/2$, then the set S that is found does not contain any source-destination pairs.

Proof.

We have: $\forall u, v \in S(r): \quad dist(u, v) \leq 2r$. Since for each SD pair (s_i, t_i), $dist(s_i, t_i) \geq 1$, the lemma follows. □

Department of Information and Computer Science

Continuous Region-Growing Process (cont.)



$d_2 >> d_1 = d_3 = d_4$ $\qquad$ $d_2 >> d_1 = d_3 = d_4 = d_5$

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Continuous Region-Growing Process (cont.)

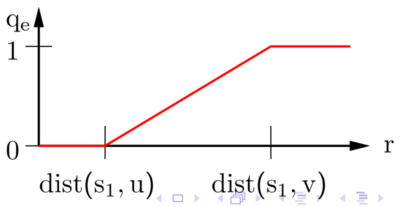
Define the weight $wt(S(r))$ of region $S(r)$ as a measure of the weight of edges between nodes of the region (recall: $c_e d_e$)

$$wt(S(r)) := wt(s_1) + \sum_{e \in E} c_e d_e q_e, \quad wt(s_1) := F/k,$$

where

$$q_e := \begin{cases} 1, & \text{if both endpoints are in } S(r) \\ \frac{r - dist(s_1,u)}{dist(s_1,v) - dist(s_1,u)}, & \text{if } e = (u,v), u \in S(r), v \notin S(r) \\ 0, & \text{if neither endpoint is in } S(r) \end{cases}$$

$$\frac{\partial q_e}{\partial r} = 1/(dist(s_1, v) - dist(s_1, u)) \geq 1/d_e \quad \forall r \in [dist(s_1, u), dist(s_1, v)]$$



HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Lemma (20.3, page 170)

Fixing $\epsilon = 2\ln(k+1)$ suffices to ensure that $c(S(r)) \leq \epsilon\, wt(S(r))$ will be encountered before $r = 1/2$ (used later for establishing the approximation guarantee).

Proof.

Assume $c(S(r)) > \epsilon\, wt(S(r))\ \forall r \in [0, 1/2]$. We have (summation over $e = (u, v), u \in S(r), v \notin S(r)$)

$$d\, wt(S(r)) = \sum_{e \in E} c_e d_e dq_e = \sum_{e \in \delta(S(r))} c_e \frac{d_e}{dist(s_1, v) - dist(s_1, u)} dr$$

$$\geq \sum_{e \in \delta(S(r))} c_e dr = c(S(r))dr > \epsilon\, wt(S(r))dr.$$

Dividing by $wt(S(r))$ and integrating over $[F/k, F + F/k]$ (substitution rule) leads to contradiction $\ln(k+1) > \frac{1}{2}\epsilon$. □

Transformation into Discrete Process

- Discrete process starts with $S = \{s_1\}$, adds vertices in increasing distance (shortest path computation at s_1)
- Definition of weight $wt^D(S)$ of region S:

$$wt^D(S) = F/k + \sum_e c_e d_e,$$

  where the sum is taken over the edges that have at least(!) one vertex in S
- Process stops when $c(S) \leq \epsilon \; wt^D(S)$, where $\epsilon = 2\ln(k+1)$
- Note: $wt^D(S) \geq wt(S) \rightarrow$ discrete process cannot terminate with larger S $\rightarrow$ S does not contain any SD pair

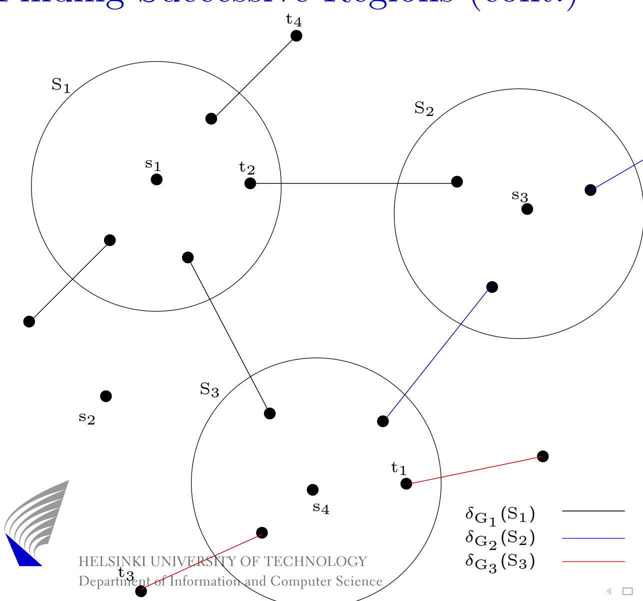
Finding Successive Regions

- Previously we only discussed one region; how does the process operate on the other regions?
- Algorithm finds sequence of regions S_i and operates on sequence of graphs G_i
- Let $G_1 = G$ and S_1 be the region found by the process when selecting any source as root of the region
- Successive graph G_2 is formed by removing vertices from S_1 and incident edges
- New root is selected among the sources of the remaining (complete!) SD pairs in G_2 and the process operates on G_2

Finding Successive Regions (cont.)

- Let the sequence of regions already found be $S_1, \ldots, S_{i-1}$
- Define G_i: graph resulting from removing vertices $\bigcup_{j=1}^{i-1} S_j$ and all edges incident to them
- If G_i does not contain any SD pair: done; otherwise pick any source of such a pair and grow a region in G_i
- All definitions (capacity, weight, etc.) defined w.r.t. G_i
- Termination condition for growing process: $c_{G_i}(S_i) \leq \epsilon \; wt_{G_i}(S_i)$
- Output $M = \bigcup_{j=1}^{l} \delta_{G_j}(S_j)$, where S_l last region found ($l \leq k$)
- Capacity $c(M) = \sum_{j=1}^{l} c_{G_j}(S_j)$ (sets $\delta_{G_j}(S_j)$ are disjoint)

Finding Successive Regions (cont.)



Note: for i = 2, 3:

$$\delta_{G_i}(S_i) \neq \delta_G(S_i)$$
$$c_{G_i}(S_i) \neq c_G(S_i)$$

Algorithm 20.4 (Minimum Multicut)

1. Find an optimal solution to relaxed LP for MinFractMulticut, obtaining edge distance labels d_e.
2. $\epsilon \leftarrow 2\ln(k+1), H \leftarrow G, M \leftarrow \emptyset$;
3. While $\exists$ source-sink pair (s_j, t_j) in H do:
   - 3.1 Grow region S with root s_j until $c_H(S) \leq \epsilon wt_H(S)$;
   - 3.2 $M \leftarrow M \cup \delta_H(S)$;
   - 3.3 $H \leftarrow H$ with vertices and incident edges of S removed;
4. Output M.

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Proving the Approximation Factor

Lemma (20.5, page 173)

The set M found is a multicut.

Proof.

We need to prove that no region contains a source-sink pair. The same argument as in the proof of Lemma 20.3 shows that the growing process in G_i terminates before $r = 1/2$. Also, the distance between any pair of vertices in region S is at most $2r < 1$ (w.r.t. G_i). Since G_i is a subgraph of G, distances in G_i cannot be smaller than in G: $dist_{G_i}(s_i, t_i) \geq dist_G(s_i, t_i) \geq 1$. □

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Proving the Approximation Factor (cont.)

Lemma (20.6, page 173)

$c(M) \leq 2\epsilon F = 4\ln(k+1)F$, *where* $c(M) = \sum_{e \in M} c_e$, $M = \bigcup_{j=1}^{l} \delta_{G_j}(S_j)$, *and* $F = \sum_{e \in E} c_e d_e$.

Proof.

At the end of iteration i we have $c_{G_i}(S_i) \leq \epsilon\, wt_{G_i}(S_i)$. Each edge of G contributes to the weight of at most one region. The total weight of all edges in G is F (by definition). Since each iteration disconnects at least one SD pair, the number of iterations is bounded by k. Therefore, the total weight attributed to source vertices is at most F. We obtain:

$$c(M) = \sum_i c_{G_i}(S_i) \leq \epsilon \left(\sum_i wt_{G_i}(S_i) \right) \leq \epsilon \left(k\frac{F}{k} + \sum_e c_e d_e \right) = 2\epsilon F$$

Proving the Approximation Factor (cont.)

Theorem (20.7, page 174)

Algorithm 20.4 achieves an approximation guarantee of $O(\log(k))$ *for the minimum multicut problem.*

Proof.

From Lemma 20.6, using the definition of F and weak duality, we obtain

$$c(M) = \sum_i c_{G_i}(S_i) \leq 4\ln(k+1) \sum_{e \in E} c_e d_e \leq 4\ln(k+1)OPT$$

□

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Approximate MaxFlow / MinCut Theorem

Corollary (20.8, page 174)

In an undirected graph with k *source-sink pairs,*

$$\max_{m/c\ flow\ F} |F| \leq \min_{multicut\ C} |C| \leq O(\log k) \left(\max_{m/c\ flow\ F} |F| \right),$$

where $|F|$ *represent the value of multicommodity flow* F*, and* $|C|$ *the value of multicut* C.

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Example 20.9

Definition: An expander graph is a graph $G = (V, E)$ in which every vertex has the same degree d and for any nonempty subset $S \subset V$,

$$|\delta(S)| > \min\{|S|, |\bar{S}|\},$$

where $\delta(S)$ denotes the edges in the cut $(S, \bar{S})$. Let H be an expander graph with $d \geq 3$, k vertices and unit edge capacities. Fixing $\alpha = \lfloor \log_d k/2 \rfloor$ ensures that for any vertex v there are at least $k/2$ vertices at distance $\alpha' \geq \alpha$ from v. For a proper selection of source-sink pairs (located at least α hops apart) each path with non-zero flow consumes $\Omega(\log k)$ total units of capacity. As the total amount of available capacity in the graph is $O(k)$, the value of the maximum multicommodity flow in H is bounded by $O(k/\log k)$. One then shows that the minimum multicut has capacity $\Omega(k)$ (using the expander graph property), thereby proving the claimed integrality gap.

Conclusions

- We have seen approximation algorithms for two version of the multicut problem
  - factor 2 for trees
  - factor $O(\log k)$ for general graphs
- For trees we also obtain an approximation algorithm for integer multicommodity flow (for general graphs no nontrivial algorithms are known)
- Application of primal-dual schema and LP rounding method (instructive?)
- Although these techniques seem to be nice, it is (at least to me) still not quite clear how to apply them in general

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science

Thanks for your attention..

Further comments or questions?

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Information and Computer Science