15-853: Algorithms in the Real World

Graph Separators
- Introduction
- Applications

Edge Separators
An edge separator:
a set of edges $E' \subseteq E$
which partitions $V$ into
$V_1$ and $V_2$
Criteria:
$|V_1|, |V_2|$ balanced
$|E'|$ is small

Vertex Separators
An vertex separator:
a set of vertices $V' \subseteq V$
which partitions $V$ into
$V_1$ and $V_2$
Criteria:
$|V_1|, |V_2|$ balanced
$|V'|$ is small

Compared with Min-cut
Min-cut: as in the min-cut, max-flow theorem.
Min-cut has no balance criteria.
Min-cut typically has a source ($s$) and sink ($t$).
Will tend to find unbalanced cuts.
Other names

Sometimes referred to as
- graph partitioning (probably more common than "graph separators")
- graph bisectors
- graph bifurcators
- balanced or normalized graph cuts

What graphs have small separators

Planar graphs: $O(n^{1/2})$ vertex separators
- 2d meshes, constant genus, excluded minors
Almost planar graphs:
- the internet, power networks, road networks
Circuits
- need to be laid out without too many crossings
Social network graphs:
- phone-call graphs, link structure of the web, citation graphs, "friends graphs"
3d-grids and meshes: $O(n^{2/3})$

What graphs don't have small separators

Hypercubes:
- $O(n)$ edge separators
- $O(n/(\log n)^{1/2})$ vertex separators
Butterfly networks:
- $O(n/\log n)$ separators?
Expander graphs:
- Graphs such that for any $U \subset V$, s.t. $|U| \leq \alpha |V|$, $|\text{neighbors}(U)| \geq \beta |U|$. ($\alpha < 1$, $\beta > 1$)
- random graphs are expanders, with high probability

It is exactly the fact that they don't have small separators that make them useful.
**Applications of Separators**

- Circuit Layout (dates back to the 60s)
- VLSI layout
- Solving linear systems (nested dissection)
- $n^{3/2}$ time for planar graphs
- Partitioning for parallel algorithms
- Approximations to certain NP hard problems
  - TSP, maximum-independent-set
- Clustering and machine learning
- Machine vision

**Available Software**

- **METIS**: U. Minnesota
- **PARTY**: University of Paderborn
- **CHACO**: Sandia national labs
- **JOSTLE**: U. Genewich
- **SCOTCH**: U. Bordeaux
- **GNU**: Popenet

**Benchmarks:**
- Graph Partitioning Archive

**More Applications of Separators**

- Out of core algorithms
- Register allocation
- Shortest Paths
- Graph compression
- Graph embeddings

**Different Balance Criteria**

- Bisectors: 50/50
- Constant fraction cuts: e.g. 1/3, 2/3

**Trading off cut size for balance:**

- Min cut criteria:
  $$\min_{V \subseteq V} \left( \frac{|V|}{|V_1||V_2|} \right)$$

- Min quotient separator:
  $$\min_{V \subseteq V} \left( \frac{|V|}{\min(|V_1||V_2|)} \right)$$

All versions are NP-hard
Other Variants of Separators

\textbf{k-Partitioning:}
Might be done with recursive partitioning, but direct solution can give better answers.

\textbf{Weighted:}
Weights on edges (cut size), vertices (balance)

\textbf{Hypergraphs:}
Each edge can have more than 2 end points common in VLSI circuits

\textbf{Multiconstraint:}
Trying to balance different values at the same time.

Asymptotics

If $S$ is a class of graphs closed under the subgraph relation, then

\textbf{Definition:} $S$ satisfies a $f(n)$ vertex-separator theorem if there are constants $\alpha < 1$ and $\beta > 0$ so that for every $G \in S$ there exists a cut set $C \subseteq V$, with

1. $|C| \leq \beta f(|G|)$ cut size
2. $|A| \leq \alpha |G|$, $|B| \leq \alpha |G|$ balance

Similar definition for edge separators.

Edge vs. Vertex separators

If a class of graphs satisfies an $f(n)$ edge-separator theorem then it satisfies an $f(n)$ vertex-separator. The other way is not true (unless degree is bounded)

Separator Trees
Separator Trees

**Theorem:** For $S$ satisfying an $(\alpha, \beta)$ $n^{1-c}$ edge separator theorem, we can generate a perfectly balanced separator tree with separator size $|C| \leq k \beta f(|G|)$.

**Proof:** by picture $|C| = \beta n^{1-c}(1 + \alpha + \alpha^2 + \ldots) = \beta n^{1-c}(1/1-\alpha)$

---

Algorithms

All are either heuristics or approximations
- Kernighan-Lin (heuristic)
- Planar graph separators (finds $O(n^{1/2})$ separators)
- Geometric separators (finds $O(n^{d-1/2})$ separators)
- Spectral (finds $O(n^{d-1/2})$ separators)
- Flow techniques (give log(n) approximations)
- Multilevel recursive bisection (heuristic, currently most practical)

---

Kernighan-Lin Heuristic

Local heuristic for edge-separators based on "hill climbing". Will most likely end in a local-minima.

Two versions:
- Original: takes $n^2$ times per step
- Fiduccia-Mattheyses: takes $n$ times per step

---

High-level description for both

Start with an initial cut that partitions the vertices into two equal size sets $V_1$ and $V_2$

Want to swap two equal sized sets $X \subseteq A$ and $Y \subseteq B$ to reduce the cut size.

Note that finding the optimal subsets $X$ and $Y$ solves the optimal separator problem, so it is NP hard. We want some heuristic that might help.
Some Terminology

$C(A,B)$: the weighted cut between $A$ and $B$

$I(v)$: the number of edges incident on $v$ that stay within the partition

$E(v)$: the number of edges incident on $v$ that go to the other partition

$D(v) = E(v) - I(v)$

$D(u,v) = D(u) + D(v) - 2w(u,v)$

- the gain for swapping $u$ and $v$

Kernighan-Lin improvement step

$KL(G,A_0,B_0)$

- $\forall u \in A_0, v \in B_0$ put $(u,v)$ in a PQ based on $D(u,v)$
- for $k = 1$ to $|V|/2$
  - $(u,v) = \max(PQ)$
  - $(A_k,B_k) = (A_{k-1},B_{k-1})$ swap $(u,v)$
  - delete $u$ and $v$ entries from $PQ$
  - update $D$ on neighbors (and $PQ$)
  - select $A_k,B_k$ with best $C_k$

Note that can take backward steps ($D(u,v)$ can be negative).

Fiduccia-Mattheyses improvement step

$FM(G,A_0,B_0)$

- $\forall u \in A_0$ put $u$ in $PQA$ based on $D(u)$
- $\forall v \in B_0$ put $v$ in $PQB$ based on $D(v)$
- for $k = 1$ to $|V|/2$
  - $u = \max(PQA)$
  - put $u$ on $B$ side and update $D$
  - $v = \max(PQB)$
  - put $v$ on $A$ side and update $D$
  - select $A_k,B_k$ with best $C_k$

A Bad Example for KL or FM

Consider following graph with initial cut given in red.

KL (or FM) will start on one side of the grid (e.g. the blue pair) and flip pairs over moving across the grid until the whole thing is flipped.

After one round the graph will look identical?
**Boundary Kernighan-Lin (or FM)**

Instead of putting all pairs (u,v) in Q (or all u and v in Q for FM), just consider the boundary vertices (i.e. vertices adjacent to a vertex in the other partition).
Note that vertices might not originally be boundaries but become boundaries.
In practice for reasonable initial cuts this can speed up KL by a large factor, but won’t necessarily find the same solution as KL.

**Performance in Practice**

In general the algorithms do very well at smoothing a cut that is approximately correct.
Works best for graphs with reasonably high degree.
Used by most separator packages either
1. to smooth final results
2. to smooth partial results during the algorithm

---

**Separators Outline**

Introduction:

Algorithms:
- Kernighan Lin
- BFS and PFS
- Multilevel
- Spectral
- Lipton Tarjan

Applications:
- Graph Compression
- Nested Dissection (solving linear systems)

**Breadth-First Search Separators**

Run BFS and as soon as you have included half the vertices return that as the partition.
Won’t necessarily be 50/50, but can arbitrarily split vertices in middle level.
Used as substep in Lipton-Tarjan planar separators.
In practice does not work well on its own.
Picking the Start Vertex
1. Try a few random starts and select best partition found
2. Start at an “extreme” point. Do an initial DFS starting at any point and select a vertex from the last level to start with.
3. If multiple extreme points, try a few of them.

Multilevel Graph Partitioning
Suggested by many researchers around the same time (early 1990s).
Packages that use it:
- METIS
- Jostle
- TSL (GNU)
- Chaco
Best packages in practice (for now), but not yet properly analyzed in terms of theory.
Mostly applied to edge separators.

Priority-First Search Separators
Prioritize the vertices based on their gain (as defined in KL) with the current set.
Search until you have half the vertices.

High-Level Algorithm Outline
MultilevelPartition(G)
If G is small, do something brute force
Else
Coarsen the graph into G’ (Coarsen)
A’,B’ = MultilevelPartition(G’)
Expand graph back to G and project the partitions A’ and B’ onto A and B
Refine the partition A,B and return result

Many choices on how to do underlined parts
**MGP as Bubble Diagram**

- **G**
- Coarsen
- Expand, Project and Refine
- "Brute Force"

**How to Coarsen**

Goal is to pick a sample $G'$ such that when we find its partition it will help us find the partition of $G$. 
**Possibilities?**

---

**Random Sampling**

- Pick a random subset of the vertices.
- Remove the unchosen vertices and their incident edges

**Random Sampling**

- Pick a random subset of the vertices.
- Remove the unchosen vertices and their incident edges
- Graph falls apart if it is not dense enough.
**Maximal Matchings**

A maximal matching is a pairing of neighbors so that no unpaired vertex can be paired with an unpaired neighbor.

The idea is to contract pairs into a single vertex.

---

**A Maximal Matching**

Can be found in linear time greedily.

---

**A side note**

Compared to a maximum matching: a pairing such that the number of covered nodes is maximum.

---

**Coarsening**
**Colapsing and Weights**

New vertices become weighted by sum of weights of their pair.
New edges \((u,v)\)
become weighted by sum of weights of multiple edges \((u,v)\)
We therefore have solve the weighted problem.

**Why care about weights?**

---

**Heuristics for finding the Matching**

- **Random**: randomly select edges.
- **Prioritized**: the edges are prioritized by weight. Visit vertices in random order, but pick highest priority edge first.
  - **Heaviest first**: Why might this be a good heuristic?
  - **Lightest first**: Why might this be a good heuristic?
- **Highly connected components**: (or heavy clique matching). Looks not only at two vertices but the connectivity of their own structure.

---

**Finding the Cut on the Coarsened Graph**

---

**Exanding and “Projecting”**
**Refining**

*E.g. by using Kernighan-Lin*

---

**After Refinement**

---

**METIS**

- **Coarsening**: "Heavy Edge" maximal matching.
- **Base case**: Priority-first search based on gain. Randomly select 4 starting points and pick best cut.
- **Smoothing**: Boundary Kernighan-Lin

Has many other options, *e.g.* Multiway separators.

---

**Separators Outline**

- **Introduction**:
- **Algorithms**:
  - Kernighan Lin
  - BFS and PFS
  - Multilevel
  - Spectral
  - Lipton Tarjan
- **Applications**:
  - Graph Compression
  - Nested Dissection (solving linear systems)
**Spectral Separators**

Based on the second eigenvector of the "Laplacian" matrix for the graph.

Let $A$ be the adjacency matrix for $G$.
Let $D$ be a diagonal matrix with degree of each vertex.
The Laplacian matrix is defined as $L = D - A$.

**Fiedler Vectors**

Find eigenvector corresponding to the second smallest eigenvalue: $L x = \lambda x$
This is called the Fiedler vector.

What is true about the first eigenvector?
Fiedler vector can be thought of as lowest frequency "mode" of vibration.

**Laplace Matrix: Example**

\[
L = \begin{pmatrix}
3 & 0 & -1 & -1 & -1 \\
0 & 1 & 0 & 0 & -1 \\
-1 & 0 & 2 & -1 & 0 \\
-1 & 0 & -1 & 3 & -1 \\
-1 & -1 & 0 & -1 & 3
\end{pmatrix}
\]

Note that each row sums to 0.

**Fiedler Vector: Example**

\[
L = \begin{pmatrix}
3 & 0 & -1 & -1 & -1 \\
0 & 1 & 0 & 0 & -1 \\
-1 & 0 & 2 & -1 & 0 \\
-1 & 0 & -1 & 3 & -1 \\
-1 & -1 & 0 & -1 & 3
\end{pmatrix}
\]

\[
x_2 = \begin{pmatrix}
-0.26 \\
0.81 \\
-0.44 \\
-0.26 \\
0.13
\end{pmatrix}
\]

$Lx_2 = 0.83x_2$
Note that each row sums to 0.
If graph is not connected, what is the second eigenvalue?
**Finding the Separator**

Sort Fiedler vector by value, and split in half.

\[ x = \begin{pmatrix} -26 \\ .81 \\ -.44 \\ -.26 \\ .13 \end{pmatrix} \]

sorted vertices: [3, 1, 4, 5, 2]

---

**Power Method**

Iterative method for finding first few eigenvectors. Every vector is a linear combination of its eigenvectors \( e_1, e_2, \ldots \)

Consider: \( p_0 = a_1 e_1 + a_2 e_2 + \ldots \)

Iterating \( p_{i+1} = Ap_i \) until it settles will give the principal eigenvector (largest magnitude eigenvalue) since \( p_i = \lambda_i^i a_1 e_1 + \lambda_i^i a_2 e_2 + \ldots \)

(Assuming all \( a_i \) are about the same magnitude)

The more spread in first two eigenvalues, the faster it will settle (related to the rapid mixing of expander graphs)

---

**The second eigenvector**

Assuming we have the principal eigenvector, after each iteration remove the component that is aligned with the principal eigenvector.

\[ n_i = A p_{i-1} \]

\[ p_i = n_i - (e_1 \cdot n_i)e_1 \] (assuming \( e_1 \) is normalized)

Now

\[ p_i = \lambda_2^i a_2 e_2 + \lambda_3^i a_3 e_3 + \ldots \]

Can use random vector for initial \( p_0 \)

---

**Power method for Laplacian**

To apply the power method we have to shift the eigenvalues, since we are interested in eigenvector with eigenvalue closest to zero.

How do we shift eigenvalues by a constant amount?

Lanczos algorithm is faster in practice if starting from scratch, but if you have an approximate solution, the power method works very well.
Multilevel Spectral

MultilevelFiedler(G)
If $G$ is small, do something brute force
Else
  Coarsen the graph into $G'$
  $e'_2 = \text{MultilevelFiedler}(G')$
  Expand graph back to $G$ and project $e'_2$ onto $e_2$
  Refine $e_2$ using power method and return

To project, you can just copy the values in location $i$ of $e'_2$ into both vertices $i$ expands into.
This idea is used by Chaco.