Deterministic and Approximation Algorithms for Graph Theoretic Problems

COMP4801: Final year Project

Omer Wasim

Department of Computer Science
The University of Hong Kong
email: omerwa90@hku.hk

April 20, 2018

Contents

Motivation: The curse of NP-completeness

- Most of the 'interesting' combinatorial optimization problems are NP-complete.
- This means there is almost no hope to find an optimal solution in polynomial time-unless $P = NP$.
- Solution: Get as close as possible to the optimal solution; in other words, compute an approximate solution.

Motivation: The curse of NP-completeness

- Most of the 'interesting' combinatorial optimization problems are NP-complete.
- This means there is almost no hope to find an optimal solution in polynomial time-unless $P = NP$.
- Solution: Get as close as possible to the optimal solution; in other words, compute an approximate solution.

Problems we studied in this project

-The written report will contain the techniques and results of the following problems we have currently surveyed (partial list).

- Minimum Multiway Cut. ([STOC] Călinescu, Karloff, and Rabani 1998)
- Maximum Cut. ([STOC] Goemans and Williamson 1995)
- Quadratic Programming. ([FOCS] Charikar and Chatziafratis 2017, Nesterov 1998,)
- Correlation Clustering. ([SODA] Swamy 2004)
- Graph Coloring. (Wigderson 1983)
- Unique Games. ([FOCS] Trevisan 2005,[STOC] Charikar, K. Makarychev, and Y. Makarychev 2006)
- Numerous variants of graph partitioning and Sparsest Cut. ([STOC]Arora, Rao, and Vazirani 2004, [KDD] Bourse, Lelarge, and Vojnovic 2014)

Our contribution

- We surveyed quite a wide array of literature in algorithms, mathematical programming and spectral graph theory.
- Major contribution is empirical evidence of various approximation algorithms on real world data sets.
- Implementation (and empirical evidence obtained) for:
  - Max-Cut SDP
  - Derandomized Max-Cut
  - LP relaxation of sparsest Cut
  - Spectral Algorithm for sparsest cut.
  - ARV algorithm for sparsest cut.

Approximation Algorithm

-Let P be a maximization problem, and let $OPT(P)$ denote the cost of optimal solution.

Definition 1 (α-approximation algorithm for P).

Algorithm A is an α-approximation algorithm for P if the cost of the solution obtained using A is *at least* $\alpha.OPT(P)$, for any instance of P.

-For a maximization problem, $\alpha \leq 1$, and the definition can be analogously extended to a minimization problem.

Approximation Algorithm

-Let P be a maximization problem, and let $OPT(P)$ denote the cost of optimal solution.

Definition 1 (α-approximation algorithm for P).

Algorithm A is an α-approximation algorithm for P if the cost of the solution obtained using A is *at least* $\alpha.OPT(P)$, for any instance of P.

-For a maximization problem, $\alpha \leq 1$, and the definition can be analogously extended to a minimization problem.

General approach to approximate discrete optimization problems

- Model an optimization problem *exactly* as an integer linear program (ILP). Note that it is NP-hard to solve an ILP.
- Relax the ILP.
- Solve the VP/LP.
- Round the solution to an integer solution.
- Bound the cost of the rounded solution.

General approach to approximate discrete optimization problems

- Model an optimization problem *exactly* as an integer linear program (ILP). Note that it is NP-hard to solve an ILP.
- Relax the ILP.
- Solve the VP/LP.
- Round the solution to an integer solution.
- Bound the cost of the rounded solution.

General approach to approximate discrete optimization problems

- Model an optimization problem *exactly* as an integer linear program (ILP). Note that it is NP-hard to solve an ILP.
- Relax the ILP.
- Solve the VP/LP.
- Round the solution to an integer solution.
- Bound the cost of the rounded solution.

General approach to approximate discrete optimization problems

- Model an optimization problem *exactly* as an integer linear program (ILP). Note that it is NP-hard to solve an ILP.
- Relax the ILP.
- Solve the VP/LP.
- Round the solution to an integer solution.
- Bound the cost of the rounded solution.

General approach to approximate discrete optimization problems

- Model an optimization problem *exactly* as an integer linear program (ILP). Note that it is NP-hard to solve an ILP.
- Relax the ILP.
- Solve the VP/LP.
- Round the solution to an integer solution.
- Bound the cost of the rounded solution.

More formally...

- Let P be a maximization problem. Let:
  - i FRAC(P): The value of the relaxed(convex) solution.
  - ii OPT(P): The value of the true combinatorial optimum.
  - iii ROUND(P): The value of the solution obtained by the rounding algorithm.
- Then, we have that ROUND(P) $\leq$OPT(P) $\leq$FRAC(P). (reverse for a min. problem)
- The goal is then to find an α s.t. ROUND(P) $\geq \alpha$FRAC(P).

More formally...

- Let P be a maximization problem. Let:
  - i FRAC(P): The value of the relaxed(convex) solution.
  - ii OPT(P): The value of the true combinatorial optimum.
  - iii ROUND(P): The value of the solution obtained by the rounding algorithm.
- Then, we have that ROUND(P) $\leq$OPT(P) $\leq$FRAC(P). (reverse for a min. problem)
- The goal is then to find an α s.t. ROUND(P) $\geq \alpha$FRAC(P).

Max-Cut Problem

Problem 2.

Given an undirected graph $G = (V, E)$, and a cost function $c : E \to \mathbb{R}^+$, partition V into 2 parts $(S, \overline(S))$ such that the total cost of edges between those parts is maximized.

- The current best known algorithm (Goemans and Williamson 1995) uses semidefinite programming and gives a 0.878-approximation. This is optimal under the Unique Games conjecture (UGC).
- Furthermore if $\alpha \geq 16/17 \approx 0.941$, then $P = NP$!

Max-Cut Problem

Problem 2.

Given an undirected graph $G = (V, E)$, and a cost function $c : E \rightarrow \mathbb{R}^+$, partition V into 2 parts $(S, \overline{(S)})$ such that the total cost of edges between those parts is maximized.

- The current best known algorithm (Goemans and Williamson 1995) uses semidefinite programming and gives a 0.878-approximation. This is optimal under the Unique Games conjecture (UGC).
- Furthermore if $\alpha \geq 16/17 \approx 0.941$, then $P = NP$!

Max-Cut Problem

Problem 2.

Given an undirected graph $G = (V, E)$, and a cost function $c : E \to \mathbb{R}^+$, partition V into 2 parts $(S, \overline(S))$ such that the total cost of edges between those parts is maximized.

- The current best known algorithm (Goemans and Williamson 1995) uses semidefinite programming and gives a 0.878-approximation. This is optimal under the Unique Games conjecture (UGC).
- Furthermore if $\alpha \geq 16/17 \approx 0.941$, then $P = NP$!

Max-Cut Problem

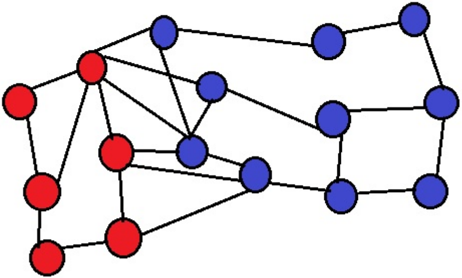

Figure 3.1: The maximum cut problem. The goal is to find a set of red and blue vertices whose union is V, such that the weight of the edges crossing between them is maximized. In this instance $c_e = 1, \forall e \in E$, so the problem reduces to finding two sets so that the number of crossing edges are maximized.

Maximum Cut

- **Goal:**Given an undirected graph $G = (V, E)$, and weights w_{ij} for all $e = (i, j) \in E$, find the cut $(S, \bar{S})$ maximizing the cost of cut edges.
- Let $y_i = 1$ if $i \in S$, and $y_i = -1$ if $i \in \bar{S}$. ILP is given as:

ILP

$$maximize \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - y_i y_j)$$

$$subject\ to y_i \in \{-1, 1\}\ for\ all\ \ i \in V$$

- Models correctly, since for any edge (i, j) in the cut $\frac{1}{2}(1 - y_i.y_j) = 1$ thus accounted once. For non-cut edge $1 - (y_i.y_j) = 0$, hence not accounted.

Maximum Cut

- **Goal:**Given an undirected graph $G = (V, E)$, and weights w_{ij} for all $e = (i, j) \in E$, find the cut $(S, \bar{S})$ maximizing the cost of cut edges.
- Let $y_i = 1$ if $i \in S$, and $y_i = -1$ if $i \in \bar{S}$. ILP is given as:

ILP

$$maximize \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - y_i y_j)$$

$$subject\ to y_i \in \{-1, 1\}\ for\ all\ \ i \in V$$

- Models correctly, since for any edge (i, j) in the cut $\frac{1}{2}(1 - y_i.y_j) = 1$ thus accounted once. For non-cut edge $1 - (y_i.y_j) = 0$, hence not accounted.

Maximum Cut

- **Goal:** Given an undirected graph $G = (V, E)$, and weights w_{ij} for all $e = (i, j) \in E$, find the cut $(S, \bar{S})$ maximizing the cost of cut edges.
- Let $y_i = 1$ if $i \in S$, and $y_i = -1$ if $i \in \bar{S}$. ILP is given as:

ILP

$$maximize \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - y_i y_j)$$

$$subject\ to y_i \in \{-1, 1\}\ for\ all\ \ i \in V$$

- Models correctly, since for any edge (i, j) in the cut $\frac{1}{2}(1 - y_i.y_j) = 1$ thus accounted once. For non-cut edge $1 - (y_i.y_j) = 0$, hence not accounted.

Maximum Cut

- **Goal:** Given an undirected graph $G = (V, E)$, and weights w_{ij} for all $e = (i, j) \in E$, find the cut $(S, \bar{S})$ maximizing the cost of cut edges.
- Let $y_i = 1$ if $i \in S$, and $y_i = -1$ if $i \in \bar{S}$. ILP is given as:

ILP

$$maximize \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - y_i y_j)$$

$$subject\ to y_i \in \{-1, 1\}\ for\ all\ \ i \in V$$

- Models correctly, since for any edge (i, j) in the cut $\frac{1}{2}(1 - y_i.y_j) = 1$ thus accounted once. For non-cut edge $1 - (y_i.y_j) = 0$, hence not accounted.

Maximum Cut (SDP relaxation)

-The corresponding SDP relaxation is:
VP

$$\begin{aligned} &\textit{maximize} \quad \frac{1}{2}\sum_{(i,j)\in E} w_{ij}(1 - v_i.v_j) \\ &\textit{subject to} \quad v_i.v_i = 1 \quad \forall i \in [n]. \end{aligned}$$

- Relaxation since given a solution $y = (y_1, y_2, ...y_n)$, one can set for all i, $v_i = (y_i, 0, ...0)$ and get a solution of the same value.
- Intuitively, **VP** optimizes over a 'greater' range of values.
- val$(VP) \geq OPT$, where OPT is an optimal value of the max-cut.

Maximum Cut (SDP relaxation)

-The corresponding SDP relaxation is:
VP

$$\begin{aligned} \textit{maximize} \quad & \frac{1}{2}\sum_{(i,j)\in E} w_{ij}(1 - v_i.v_j) \\ \textit{subject to} \quad & v_i.v_i = 1 \quad \forall i \in [n]. \end{aligned}$$

- Relaxation since given a solution $y = (y_1, y_2, ...y_n)$, one can set for all i, $v_i = (y_i, 0, ...0)$ and get a solution of the same value.
- Intuitively, **VP** optimizes over a 'greater' range of values.
- val$(VP) \geq OPT$, where OPT is an optimal value of the max-cut.

Maximum Cut (SDP relaxation)

-The corresponding SDP relaxation is:

VP

$$\text{maximize} \quad \frac{1}{2} \sum_{(i,j) \in E} w_{ij}(1 - v_i . v_j)$$

$$\text{subject to} \quad v_i . v_i = 1 \quad \forall i \in [n].$$

- Relaxation since given a solution $y = (y_1, y_2, ... y_n)$, one can set for all i, $v_i = (y_i, 0, ... 0)$ and get a solution of the same value.
- Intuitively, **VP** optimizes over a ‘greater’ range of values.
- val(VP) $\geq OPT$, where OPT is an optimal value of the max-cut.

Rounding Algorithm using random hyperplane

Randomized Rounding for Max-Cut:

i Pick a random vector $r = (r_1, ..., r_n) \in \mathbb{R}^n$ s.t. $\forall i$, $r_i \sim N(0,1)$.

ii For all $i \in V$, put i in S if $r.v_i \geq 0$, and in $\bar{S}$ otherwise.

-Can be solved with an additive error of ϵ, time polynomial in n and $log(1/\epsilon)$.

Rounding Algorithm using random hyperplane

Randomized Rounding for Max-Cut:

i Pick a random vector $r = (r_1, ..., r_n) \in \mathbb{R}^n$ s.t. $\forall i$, $r_i \sim N(0, 1)$.

ii For all $i \in V$, put i in S if $r.v_i \geq 0$, and in $\bar{S}$ otherwise.

-Can be solved with an additive error of ϵ, time polynomial in n and $log(1/\epsilon)$.

Rounding Algorithm using random hyperplane

Randomized Rounding for Max-Cut:

i Pick a random vector $r = (r_1, ..., r_n) \in \mathbb{R}^n$ s.t. $\forall i$, $r_i \sim N(0, 1)$.

ii For all $i \in V$, put i in S if $r.v_i \geq 0$, and in $\bar{S}$ otherwise.

-Can be solved with an additive error of ϵ, time polynomial in n and $log(1/\epsilon)$.

Rounding Algorithm(Analysis)

Fact 3.

The normalization of r, i.e. $\frac{r}{\|r\|}$ is uniformly distributed over the unit sphere in R^n. Moreover, for any 2 vectors x, y, the distribution of $r.x$ and $r.y$ are independent iff x, y are orthogonal.

Theorem 4.

The probability that any edge (i, j) is in the cut is $\frac{1}{\pi} arccos(v_i.v_j)$.

Proof Omitted!

Randomized Rounding (illustration)

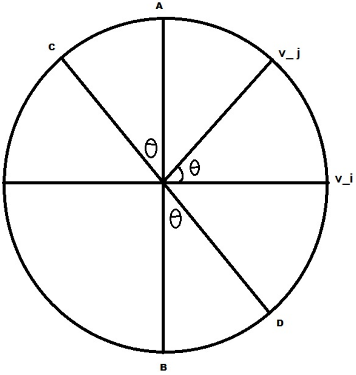

Figure 3.2: The inner products $r_p.v_i$ and $r_p.v_j$ have opposite signs iff r lies in the two sectors subtending angle θ.

SDP at work

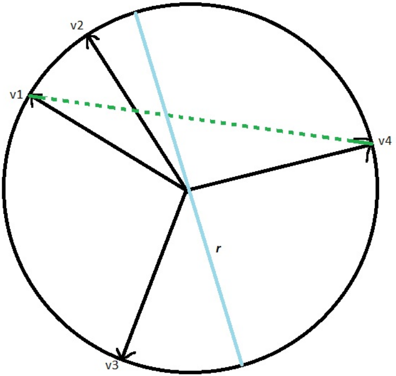

Figure 3.3: Vertices connected by an edge in G are likely to be separated far apart. Random hyperplane cuts such an edge whp.

Analysis

Fact 5.

$\frac{1}{\pi} arccos(x) \geq 0.878.\frac{1}{2}(1-x)$ *for* $x \in [-1,1]$.

-Let X_e be the r.v which is 1 if e is in the cut and 0 otherwise. Then,

$$E[\sum_{e=(i,j)\in E} c_e X_e] = \sum_{e=(i,j)\in E} c_e.Pr[X_e = 1]$$
$$= \sum_{e=(i,j)\in E} c_e \frac{1}{\pi} arccos(v_i.v_j)$$
$$\geq \sum_{e=(i,j)\in E} c_e.0.878\frac{1}{2}(1 - v_i.v_j)$$
$$= 0.878.val(VP) \geq 0.878.OPT.$$

-Thus, the expected value of the cut is within 0.878 of the optimal.

Analysis

Fact 5.

$\frac{1}{\pi}arccos(x) \geq 0.878.\frac{1}{2}(1-x)$ *for* $x \in [-1,1]$.

-Let X_e be the r.v which is 1 if e is in the cut and 0 otherwise. Then,

$$E[\sum_{e=(i,j)\in E} c_e X_e] = \sum_{e=(i,j)\in E} c_e.Pr[X_e = 1]$$
$$= \sum_{e=(i,j)\in E} c_e \frac{1}{\pi} arccos(v_i.v_j)$$
$$\geq \sum_{e=(i,j)\in E} c_e.0.878\frac{1}{2}(1 - v_i.v_j)$$
$$= 0.878.val(VP) \geq 0.878.OPT.$$

-Thus, the expected value of the cut is within 0.878 of the optimal.

Analysis

Fact 5.

$\frac{1}{\pi} arccos(x) \geq 0.878.\frac{1}{2}(1-x)$ *for* $x \in [-1, 1]$.

-Let X_e be the r.v which is 1 if e is in the cut and 0 otherwise. Then,

$$E[\sum_{e=(i,j)\in E} c_e X_e] = \sum_{e=(i,j)\in E} c_e.Pr[X_e = 1]$$
$$= \sum_{e=(i,j)\in E} c_e \frac{1}{\pi} arccos(v_i.v_j)$$
$$\geq \sum_{e=(i,j)\in E} c_e.0.878\frac{1}{2}(1 - v_i.v_j)$$
$$= 0.878.val(VP) \geq 0.878.OPT.$$

-Thus, the expected value of the cut is within 0.878 of the optimal.

Analysis

Fact 5.

$\frac{1}{\pi} arccos(x) \geq 0.878.\frac{1}{2}(1-x)$ *for* $x \in [-1, 1]$.

-Let X_e be the r.v which is 1 if e is in the cut and 0 otherwise. Then,

$$\begin{aligned} E[\sum_{e=(i,j)\in E} c_e X_e] &= \sum_{e=(i,j)\in E} c_e.Pr[X_e = 1] \\ &= \sum_{e=(i,j)\in E} c_e \frac{1}{\pi} arccos(v_i.v_j) \\ &\geq \sum_{e=(i,j)\in E} c_e.0.878\frac{1}{2}(1 - v_i.v_j) \\ &= 0.878.val(VP) \geq 0.878.OPT. \end{aligned}$$

-Thus, the expected value of the cut is within 0.878 of the optimal.

A deterministic $\frac{1}{2}$-approximation Algorithm

Derandomized Max-Cut (G=(V,E)):

1: $C = \{v_1\}$ ▷ C denotes the cut which maximizes $|E(C, \bar{C})|$
2: **for** $i = 2, ..., |V|$ **do**
3: **if** cut-edges$(C, v_i) \leq$ *degree*(v_i) **then**
4: $C = C \bigcup \{v_i\}$. ▷ cut-edges(C, v_i) denotes the number of cut edges incident to v_i and C.
5: Return C

Empirical Results

Setup:

1) Run the Max-Cut SDP and Derandomized Max-Cut on each data set and compare the performance.

2) For development, use MATLAB and CVX. Data sets from many types of real world networks.

3) Due to memory constraints, size restricted to 230 nodes.

3) For algorithm A, on an input I, compute the following normalized ratio, β_A :

$$\beta_A^I = \frac{\#\text{cut edges output by A on } I}{|E|}$$

Empirical Results

Setup:

1) Run the Max-Cut SDP and Derandomized Max-Cut on each data set and compare the performance.

2) For development, use MATLAB and CVX. Data sets from many types of real world networks.

3) Due to memory constraints, size restricted to 230 nodes.

3) For algorithm A, on an input I, compute the following normalized ratio, β_A :

$$\beta_A^I = \frac{\#\text{cut edges output by A on } I}{|E|}$$

Empirical Results

- Let B_{DR} and B_{SDP} denote the values computed.
- It follows that $B_{DR} \in [0.5, 1]$.
- If OPT(MC) is an algorithm to compute Max-Cut exactly, then:

$$0.5 \leq \beta_{DR} \leq \beta_{OPT(MC)} \leq 1 \ \forall I.$$

- Intuitively, if β_{SDP} is not too small as compared to β_{DR} the Max-Cut SDP algorithm is 'good enough'.
- Note that $1 - \beta_{DR}$ is an upper bound of the difference between $\beta_{OPT(MC)}$ and β_{DR}.

Empirical Results

- Let B_{DR} and B_{SDP} denote the values computed.
- It follows that $B_{DR} \in [0.5, 1]$.
- If OPT(MC) is an algorithm to compute Max-Cut exactly, then:

$$0.5 \leq \beta_{DR} \leq \beta_{OPT(MC)} \leq 1 \ \forall I.$$

- Intuitively, if β_{SDP} is not too small as compared to β_{DR} the Max-Cut SDP algorithm is 'good enough'.
- Note that $1 - \beta_{DR}$ is an upper bound of the difference between $\beta_{OPT(MC)}$ and β_{DR}.

Empirical Results

- Let B_{DR} and B_{SDP} denote the values computed.
- It follows that $B_{DR} \in [0.5, 1]$.
- If OPT(MC) is an algorithm to compute Max-Cut exactly, then:

$$0.5 \leq \beta_{DR} \leq \beta_{OPT(MC)} \leq 1 \ \forall I.$$

- Intuitively, if β_{SDP} is not too small as compared to β_{DR} the Max-Cut SDP algorithm is 'good enough'.
- Note that $1 - \beta_{DR}$ is an upper bound of the difference between $\beta_{OPT(MC)}$ and β_{DR}.

Empirical Results

- Let B_{DR} and B_{SDP} denote the values computed.
- It follows that $B_{DR} \in [0.5, 1]$.
- If OPT(MC) is an algorithm to compute Max-Cut exactly, then:

$$0.5 \leq \beta_{DR} \leq \beta_{OPT(MC)} \leq 1 \ \forall I.$$

- Intuitively, if β_{SDP} is not too small as compared to β_{DR} the Max-Cut SDP algorithm is 'good enough'.
- Note that $1 - \beta_{DR}$ is an upper bound of the difference between $\beta_{OPT(MC)}$ and β_{DR}.

Empirical Results

- Let B_{DR} and B_{SDP} denote the values computed.
- It follows that $B_{DR} \in [0.5, 1]$.
- If OPT(MC) is an algorithm to compute Max-Cut exactly, then:
$$0.5 \leq \beta_{DR} \leq \beta_{OPT(MC)} \leq 1 \ \forall I.$$
- Intuitively, if β_{SDP} is not too small as compared to β_{DR} the Max-Cut SDP algorithm is 'good enough'.
- Note that $1 - \beta_{DR}$ is an upper bound of the difference between $\beta_{OPT(MC)}$ and β_{DR}.

Key results

- The average difference, $\beta_{DR} - \beta_{SDP}$ across 46 graph data sets was ≈ 0.17.
- On one data set, $\beta_{SDP} > \beta_{DR} \implies$ on adversarial inputs, Max-Cut SDP might fare better.
- Note that Max-Cut SDP implemented still gives a randomized approximation. Possible to de-randomize to get deterministic 0.878-approximation.
- Overall, the derandomized version is more efficient in practice.

Key results

- The average difference, $\beta_{DR} - \beta_{SDP}$ across 46 graph data sets was ≈ 0.17.
- On one data set, $\beta_{SDP} > \beta_{DR} \implies$ on adversarial inputs, Max-Cut SDP might fare better.
- Note that Max-Cut SDP implemented still gives a randomized approximation. Possible to de-randomize to get deterministic 0.878-approximation.
- Overall, the derandomized version is more efficient in practice.

Key results

- The average difference, $\beta_{DR} - \beta_{SDP}$ across 46 graph data sets was ≈ 0.17.
- On one data set, $\beta_{SDP} > \beta_{DR} \implies$ on adversarial inputs, Max-Cut SDP might fare better.
- Note that Max-Cut SDP implemented still gives a randomized approximation. Possible to de-randomize to get deterministic 0.878-approximation.
- Overall, the derandomized version is more efficient in practice.

Key results

- The average difference, $\beta_{DR} - \beta_{SDP}$ across 46 graph data sets was ≈ 0.17.
- On one data set, $\beta_{SDP} > \beta_{DR} \implies$ on adversarial inputs, Max-Cut SDP might fare better.
- Note that Max-Cut SDP implemented still gives a randomized approximation. Possible to de-randomize to get deterministic 0.878-approximation.
- Overall, the derandomized version is more efficient in practice.

Plot of β_{DR}, β_{SDP}

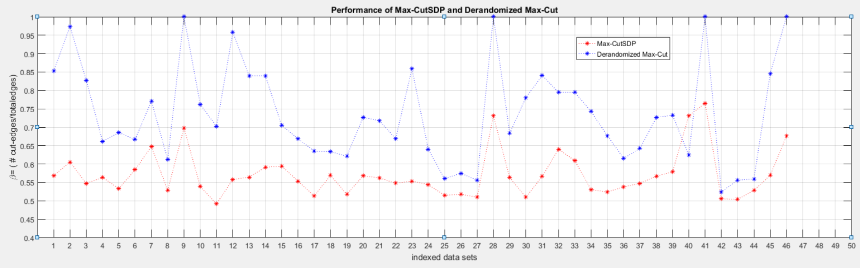

Figure 3.4: As can be seen, Derandomized Max-Cut fares well on nearly all instances. However,for most inputs the difference between the values is small. As claimed, $\beta_{DR} \geq 0.5$

Plot of $\beta_{DR} - \beta_{SDP}$

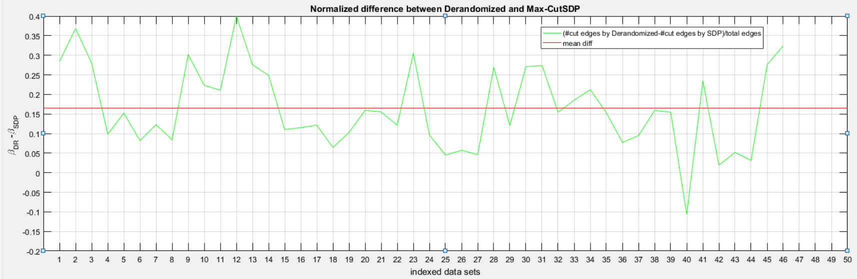

Figure 3.5: The mean difference is about 0.17 shown as the horizontal line, while the highest difference is about 0.4.

The Uniform Sparsest Cut Problem

Problem 6 (Uniform Sparsest Cut).

Given an undirected graph $G = (V, E)$, costs $c_e \, \forall e \in E$, and a single unit demand between all $s, t \in V$, find a set of vertices minimizing

$$\rho(S) = \frac{\sum_{e \in \delta(S)} c_e}{|S||V - S|}.$$

If all costs $c_e = 0$, then the goal is to minimize (over all $S \subseteq V$:

$$\rho(S) = \frac{|E(S, \bar{S})|}{|S||V - S|}$$

Finding sparse cuts in a graph

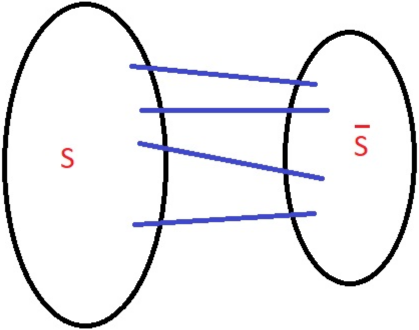

Figure 4.1: Finding a sparse cut is equivalent to finding a set S minimizing cut edges while ensuring 'balance'.

Relation to edge-expansion

- **Claim:** The uniform sparsest cut can be used to approximate the edge expansion within factor 2.
- The edge expansion of a cut $S \subseteq V$ for $|S| \leq n/2$ is $\phi(S) = \dfrac{\delta(S)}{|S|}$.
- For graph G, $\phi(G) = \min_{S \subseteq V, |S| \leq n/2} \phi(S)$. Since $\frac{n}{2} \leq |V - S| \leq n$, the claim follows.

  -We briefly discuss the 3 algorithms (which give increasingly better approximations).

Relation to edge-expansion

- **Claim:** The uniform sparsest cut can be used to approximate the edge expansion within factor 2.
- The edge expansion of a cut $S \subseteq V$ for $|S| \leq n/2$ is $\phi(S) = \dfrac{\delta(S)}{|S|}$.
- For graph G, $\phi(G) = \min_{S \subseteq V, |S| \leq n/2} \phi(S)$. Since $\frac{n}{2} \leq |V - S| \leq n$, the claim follows.

-We briefly discuss the 3 algorithms (which give increasingly better approximations).

Relation to edge-expansion

- **Claim:** The uniform sparsest cut can be used to approximate the edge expansion within factor 2.
- The edge expansion of a cut $S \subseteq V$ for $|S| \leq n/2$ is $\phi(S) = \dfrac{\delta(S)}{|S|}$.
- For graph G, $\phi(G)=\min_{S \subseteq V, |S| \leq n/2} \phi(S)$. Since $\frac{n}{2} \leq |V - S| \leq n$, the claim follows.

  -We briefly discuss the 3 algorithms (which give increasingly better approximations).

The spectral approach

- Use the normalized laplacian matrix, L to compute the second eigenvalue and eigenvector. L is defined to be:
$$L = I - D^{-1/2}AD^{-1/2}$$
where A is the adjacency matrix and $D_{ii} = d_i \; \forall i \in V$, d_i is the degree of i in V.

- Let $\Phi(G)$ be the graph conductance defined as:
$$\Phi(G) = \min_{S:|S|\leq|V|/2} \Phi(S) = \min_{S:vol(S)\leq vol(G)/2} \frac{|E(S,\bar{S})|}{vol(S)} = \min_{S:vol(S)\leq vol(G)/2} \frac{|E(S,\bar{S})|}{\sum_{v\in S} d_v}$$

- If λ_2 is the second eigenvalue of L, then from Cheeger's inequalities:
$$\frac{\lambda_2}{2} \leq \Phi(G) \leq \sqrt{2\lambda_2}.$$

The spectral approach

- Use the normalized laplacian matrix, L to compute the second eigenvalue and eigenvector. L is defined to be:

$$L = I - D^{-1/2}AD^{-1/2}$$

where A is the adjacency matrix and $D_{ii} = d_i \; \forall i \in V$, d_i is the degree of i in V.

- Let $\Phi(G)$ be the graph conductance defined as:

$$\Phi(G) = \min_{S:|S|\leq|V|/2} \Phi(S) = \min_{S:vol(S)\leq vol(G)/2} \frac{|E(S,\bar{S})|}{vol(S)} = \min_{S:vol(S)\leq vol(G)/2} \frac{|E(S,\bar{S})|}{\sum_{v\in S} d_v}$$

- If λ_2 is the second eigenvalue of L, then from Cheeger's inequalities:

$$\frac{\lambda_2}{2} \leq \Phi(G) \leq \sqrt{2\lambda_2}.$$

The spectral approach

- Use the normalized laplacian matrix, L to compute the second eigenvalue and eigenvector. L is defined to be:
$$L = I - D^{-1/2} A D^{-1/2}$$
where A is the adjacency matrix and $D_{ii} = d_i \ \forall i \in V$, d_i is the degree of i in V.

- Let $\Phi(G)$ be the graph conductance defined as:
$$\Phi(G) = \min_{S:|S| \leq |V|/2} \Phi(S) = \min_{S:vol(S) \leq vol(G)/2} \frac{|E(S,\bar{S})|}{vol(S)} = \min_{S:vol(S) \leq vol(G)/2} \frac{|E(S,\bar{S})|}{\sum_{v \in S} d_v}$$

- If λ_2 is the second eigenvalue of L, then from Cheeger's inequalities:
$$\frac{\lambda_2}{2} \leq \Phi(G) \leq \sqrt{2\lambda_2}.$$

Fiedler's spectral algorithm

- Given the eigenvector x_2 corresponding to λ_2, the second eigenvalue, finds a cut $(S, \bar{S})$ such that:
$$\min\{\Phi(S), \Phi(\bar{S})\} \leq \sqrt{2\lambda_2} \leq 2\sqrt{\Phi(G)},$$
- Time complexity: $O(|E| + |V| log|V|)$.
- **Spectral algorithm ($G = (V, E), x_2$):**
  1: Sort $v \in V$ according to the entries in x_2.
  2: Output the cut which minimizes $\Phi(v_1, .., v_k)$ for $k = 1, .., n-1$.
- Good approximation for graphs of constant expansion.
- Since $\phi(G), \Phi(G)$ and $\rho(G)$ are inter-reducible, the algorithm gives a good 'sparse' cut.

Fiedler's spectral algorithm

- Given the eigenvector x_2 corresponding to λ_2, the second eigenvalue, finds a cut $(S, \bar{S})$ such that:

$$\min\{\Phi(S), \Phi(\bar{S})\} \leq \sqrt{2\lambda_2} \leq 2\sqrt{\Phi(G)},$$

- Time complexity: $O(|E| + |V| log |V|)$.

- **Spectral algorithm** $(G = (V, E), x_2)$**:**
  1: Sort $v \in V$ according to the entries in x_2.
  2: Output the cut which minimizes $\Phi(v_1, .., v_k)$ for $k = 1, .., n - 1$.

- Good approximation for graphs of constant expansion.

- Since $\phi(G), \Phi(G)$ and $\rho(G)$ are inter-reducible, the algorithm gives a good 'sparse' cut.

Fiedler's spectral algorithm

- Given the eigenvector x_2 corresponding to λ_2, the second eigenvalue, finds a cut $(S, \bar{S})$ such that:

$$\min\{\Phi(S), \Phi(\bar{S})\} \leq \sqrt{2\lambda_2} \leq 2\sqrt{\Phi(G)},$$

- Time complexity: $O(|E| + |V| log |V|)$.
- **Spectral algorithm ($G = (V, E), x_2$):**
  1: Sort $v \in V$ according to the entries in x_2.
  2: Output the cut which minimizes $\Phi(v_1, .., v_k)$ for $k = 1, .., n-1$.
- Good approximation for graphs of constant expansion.
- Since $\phi(G), \Phi(G)$ and $\rho(G)$ are inter-reducible, the algorithm gives a good 'sparse' cut.

Fiedler's spectral algorithm

- Given the eigenvector x_2 corresponding to λ_2, the second eigenvalue, finds a cut $(S, \bar{S})$ such that:

$$\min\{\Phi(S), \Phi(\bar{S})\} \leq \sqrt{2\lambda_2} \leq 2\sqrt{\Phi(G)},$$

- Time complexity: $O(|E| + |V| log |V|)$.
- **Spectral algorithm ($G = (V, E), x_2$):**
  1. Sort $v \in V$ according to the entries in x_2.
  2. Output the cut which minimizes $\Phi(v_1, .., v_k)$ for $k = 1, .., n - 1$.
- Good approximation for graphs of constant expansion.
- Since $\phi(G), \Phi(G)$ and $\rho(G)$ are inter-reducible, the algorithm gives a good 'sparse' cut.

Leighton-Rao's LP relaxation

- Uses a linear programming relaxation and relaxes indicator functions to arbitrary semimetrics.
- If $1_S(u) = 1$ whenever $u \in S$, then
$$\rho(S) = \frac{|E(S, \bar{S})|}{|S||\bar{S}|} = \frac{\sum_{(u,v)\in E} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}{\sum_{u \in S, v \in \bar{S}} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}$$
- Note that if $d_S(u, v) = |1_S(u) - 1_S(v)|$, then d is a semimetric.
- LR: Relax the indicator function to arbitrary semimemtrics.
$$\min_{d:d \text{ is a semimetric}} \frac{\sum_{(u,v)\in E} d(u,v)}{\sum_{u,v \in V} d(u,v)}$$
- The denominator in LP is normalized to 1.

Leighton-Rao's LP relaxation

- Uses a linear programming relaxation and relaxes indicator functions to arbitrary semimetrics.

- If $1_S(u) = 1$ whenever $u \in S$, then

$$\rho(S) = \frac{|E(S,\bar{S})|}{|S||\bar{S}|} = \frac{\sum_{(u,v)\in E} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}{\sum_{u\in S, v\in \bar{S}} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}$$

- Note that if $d_S(u,v) = |1_S(u) - 1_S(v)|$, then d is a semimetric.

- LR: Relax the indicator function to arbitrary semimemtrics.

$$\min_{d:d \text{ is a semimetric}} \frac{\sum_{(u,v)\in E} d(u,v)}{\sum_{u,v\in V} d(u,v)}$$

- The denominator in LP is normalized to 1.

Leighton-Rao's LP relaxation

- Uses a linear programming relaxation and relaxes indicator functions to arbitrary semimetrics.
- If $1_S(u) = 1$ whenever $u \in S$, then
$$\rho(S) = \frac{|E(S,\bar{S})|}{|S||\bar{S}|} = \frac{\sum_{(u,v)\in E} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}{\sum_{u\in S, v\in \bar{S}} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}$$
- Note that if $d_S(u,v) = |1_S(u) - 1_S(v)|$, then d is a semimetric.
- LR: Relax the indicator function to arbitrary semimemtrics.
$$\min_{d:d \text{ is a semimetric}} \frac{\sum_{(u,v)\in E} d(u,v)}{\sum_{u,v\in V} d(u,v)}$$
- The denominator in LP is normalized to 1.

Leighton-Rao's LP relaxation

- Uses a linear programming relaxation and relaxes indicator functions to arbitrary semimetrics.
- If $1_S(u) = 1$ whenever $u \in S$, then
$$\rho(S) = \frac{|E(S,\bar{S})|}{|S||\bar{S}|} = \frac{\sum_{(u,v)\in E} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}{\sum_{u\in S, v\in \bar{S}} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}$$
- Note that if $d_S(u,v) = |1_S(u) - 1_S(v)|$, then d is a semimetric.
- LR: Relax the indicator function to arbitrary semimemtrics.
$$\min_{d:d \text{ is a semimetric}} \frac{\sum_{(u,v)\in E} d(u,v)}{\sum_{u,v\in V} d(u,v)}$$
- The denominator in LP is normalized to 1.

Leighton-Rao's LP relaxation

- Uses a linear programming relaxation and relaxes indicator functions to arbitrary semimetrics.
- If $1_S(u) = 1$ whenever $u \in S$, then
$$\rho(S) = \frac{|E(S, \bar{S})|}{|S||\bar{S}|} = \frac{\sum_{(u,v)\in E} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}{\sum_{u\in S, v\in \bar{S}} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}$$
- Note that if $d_S(u, v) = |1_S(u) - 1_S(v)|$, then d is a semimetric.
- LR: Relax the indicator function to arbitrary semimemtrics.
$$\min_{d: d \text{ is a semimetric}} \frac{\sum_{(u,v)\in E} d(u, v)}{\sum_{u,v\in V} d(u, v)}$$
- The denominator in LP is normalized to 1.

Leighton-Rao's LP relaxation

- **LP**

$$\begin{aligned} minimize \quad & \sum_{(u,v)\in E} d_{uv} \\ subject\ to \quad & \sum_{u,v\in S} d_{uv} = 1 \\ & d_{uv} \leq d_{uw} + d_{w,v} \ \ \forall u, v, w \in V \\ & d_{u,v} \geq 0 \ \ \forall u, v \in V \end{aligned}$$

Leighton-Rao's $O(log\ n)$ approximation

Theorem 7 (On the equivalence, and existence of f).

$$\rho(G) = \min_S \frac{\sum_{(u,v)\in E} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}{\sum_{u\in S, v\in \bar{S}} |\mathbf{1}_S(u) - \mathbf{1}_S(v)|}$$

$$= \inf_{m, f: V\to\mathbb{R}^n} \frac{\sum_{u,v\in E} \|f(u) - f(v)\|_1}{\sum_{u\in S, v\in \bar{S}} \|f(u) - f(v)\|_1}$$

for some f, m.

-Now apply Bourgain's theorem d on the semimetric to find f, m.

Leighton-Rao's $O(log\ n)$ approximation

Theorem 8 ($O(log\ n)$ approximation).

Given distances d_{uv} for all $u, v \in V$, one can find an embedding in polynomial time $f : V \to \mathbb{R}^m$ such that, with high probability for all $u, v \in V$

$$\|f(u) - f(v)\|_1 \leq d_{uv} \leq O(\log n) \|f(u) - f(v)\|_1$$

yielding an $O(log\ n)$ approximation guarantee for the USC.

-Uses the constructive proof of Bourgain's theorem to generate $m = log^2(n)$ subsets to construct a Frechet embedding.

-Outputs the sparsest cut by sorting vertices along the minimum dimension of their l_1 distance.

ARV Algorithm (STOC '04)

- Major contribution: Algorithm to generate well separated sets.
- The algorithm is tight for the n dimensional hypercube within constant factor.
- Uses and SDP relaxation combined with the triangle inequality for l_2^2 norm.

SDP relaxation for Sparsest Cut

- **SDP Relaxation:**

$$\begin{aligned} &\textit{minimize} \ \frac{1}{n^2} \sum_{e=(i,j)\in E} c_e \left\| v_i - v_j \right\|^2 \\ &\textit{subject to} \ \sum_{i,j\in V: i\neq j} \left\| v_i - v_j \right\|^2 = n^2 \\ &\qquad \left\| v_i - v_j \right\|^2 \leq \left\| v_i - v_k \right\|^2 + \left\| v_k - v_j \right\|^2 \ \forall i,j,k \in V \\ &\qquad v_i \in \mathbb{R}^n \ \forall i \in V. \end{aligned}$$

Ideas behind the Algorithm:

- Map the vertices to points on the unit sphere in $\mathbb{R}^n$ minimizing the sum of the squares of the edge lengths while restricting the the square distance between the average pair of points to a constant.
- Given such points, one can find almost disjoint 'antipodal' sets, L and R which are 'well separated'.
- Choose a random distance and output the sparsest cut by considering all points within distance r of L.
- Uses results from measure concentration. Proofs are long.

Preliminaries

Definition 9 (Closed ball around i).

Let $i \in V$. Then the ball of radius r around v is given by $B(i,r) = \{j \in V : d(i,j) \leq r\}$. Note that $d(i,j) = \|v_i - v_j\|^2$. Also known as the l_2^2 metric, the square of the usual Euclidean metric.

Definition 10 (Δ-Separated Sets).

Let $d(i,S) = \min_{j \in S} d(i,j)$. Sets S, T are Δ-separated if $\forall i,j \in \|v_i - v_j\|^2 \geq \Delta$.

Definition 11 ($\alpha-$large sets).

Sets L, R are $\alpha-$large if $|L| \geq \alpha.n$, $|R| \geq \alpha.n$.

Preliminaries

Definition 9 (Closed ball around i).

Let $i \in V$. Then the ball of radius r around v is given by $B(i, r) = \{j \in V : d(i, j) \leq r\}$. Note that $d(i, j) = \|v_i - v_j\|^2$. Also known as the l_2^2 metric, the square of the usual Euclidean metric.

Definition 10 (Δ-Separated Sets).

Let $d(i, S) = \min_{j \in S} d(i, j)$. Sets S, T are Δ-separated if $\forall i, j \in \|v_i - v_j\|^2 \geq \Delta$.

Definition 11 ($\alpha-$large sets).

Sets L, R are $\alpha-$large if $|L| \geq \alpha.n$, $|R| \geq \alpha.n$.

Preliminaries

Definition 9 (Closed ball around i).

Let $i \in V$. Then the ball of radius r around v is given by $B(i,r) = \{j \in V : d(i,j) \le r\}$. Note that $d(i,j) = \|v_i - v_j\|^2$. Also known as the l_2^2 metric, the square of the usual Euclidean metric.

Definition 10 (Δ-Separated Sets).

Let $d(i,S) = \min_{j \in S} d(i,j)$. Sets S, T are Δ-separated if $\forall i,j \in \|v_i - v_j\|^2 \ge \Delta$.

Definition 11 ($\alpha-$large sets).

Sets L, R are $\alpha-$large if $|L| \ge \alpha.n$, $|R| \ge \alpha.n$.

Algorithm to generate well-separated sets

Algorithm (Sparsest Cut via Fat-Hyperplane Rounding)

if there is an $i \in V$ st $|B(i, 1/4)| \geq n/4$ **then**
  $L' = B(i, 1/4)$
else
  Pick $o \in V$ which maximizes $|B(o, 4)|$
  Pick a random vector r.
  Let $L = \{i \in V : (v_i - v_o).r \geq \sigma\}$, $R = \{i \in V : (v_i - v_o).r \leq -\sigma\}$
  Let $L' = L, R' = R$
  while there exists $i \in L', j \in R'$ st $d(i, j) \leq \Delta$ **do**
    Remove i, j from L', R' resp.
Sort $i \in V$ in non-dec order of $d(i, L')$ to get $i_1, .., i_n$.
Return $S_k = \{i_1, i_2, .., i_k\}$ which minimizes $p(S_k)$, $1 \leq k \leq n$.

Key Theorems(without proof)

Theorem 12 (Large-enough L, R).

If there is no $i \in V$ st $|B(i, 1/4)| \geq \frac{n}{4}$, then with constant probability, L, R are α-large for some constant α.

Theorem 13 (Large-enough and well separated L', R').

If L, R are α-large, then with constant probability, L', R' are $\alpha/2$ large, and Δ-separated where $\Delta = C/\sqrt{\log n}$ for some C.

Theorem 14 (Leading to proof of $O(\sqrt{\log n})$ guarantee).

A cut S st $L \subseteq S \subseteq V - R$ can be found st.

$$\rho(S) \leq \frac{\sum_{e=(i,j)\in E} c_e \|v_i - v_j\|^2}{\sum_{i \in L, j \in R} \|v_i - v_j\|^2}.$$

Key Theorems(without proof)

Theorem 12 (Large-enough L, R).

If there is no $i \in V$ st $|B(i, 1/4)| \geq \frac{n}{4}$, then with constant probability, L, R are α-large for some constant α.

Theorem 13 (Large-enough and well separated L', R').

If L, R are α-large, then with constant probability, L', R' are $\alpha/2$ large, and Δ-separated where $\Delta = C/\sqrt{\log n}$ for some C.

Theorem 14 (Leading to proof of $O(\sqrt{\log n})$ guarantee).

A cut S st $L \subseteq S \subseteq V - R$ can be found st.

$$\rho(S) \leq \frac{\sum_{e=(i,j)\in E} c_e \|v_i - v_j\|^2}{\sum_{i\in L, j\in R} \|v_i - v_j\|^2}.$$

Key Theorems(without proof)

Theorem 12 (Large-enough L, R).

If there is no $i \in V$ st $|B(i, 1/4)| \geq \frac{n}{4}$, then with constant probability, L, R are α-large for some constant α.

Theorem 13 (Large-enough and well separated L', R').

If L, R are α-large, then with constant probability, L', R' are $\alpha/2$ large, and Δ-separated where $\Delta = C/\sqrt{\log n}$ for some C.

Theorem 14 (Leading to proof of $O(\sqrt{\log n})$ guarantee).

A cut S st $L \subseteq S \subseteq V - R$ can be found st.

$$\rho(S) \leq \frac{\sum_{e=(i,j)\in E} c_e \left\| v_i - v_j \right\|^2}{\sum_{i \in L, j \in R} \left\| v_i - v_j \right\|^2}.$$

Proof sketch of $O(\sqrt{log\ n})$ guarantee

From theorem 17, once such a cut S has been found, we have

Proof of $O(\sqrt{log\ n})$ guarantee

Note that,

$$\sum_{i,j\in V:i\neq j} \|v_i - v_j\|^2 \geq \sum_{i\in L, j\in R} \|v_i - v_j\|^2$$
$$\geq \Omega(n^2/\sqrt{log\ n}).$$

Then,

$$\rho(S) \leq O(\sqrt{log\ n})\frac{1}{n^2}\sum_{e=(i,j)\in E} \|v_i - v_j\|^2$$
$$\leq O(\sqrt{log\ n}).OPT.$$

The last inequality follows from the SDP relaxation.

Proof sketch of $O(\sqrt{\log n})$ guarantee

From theorem 17, once such a cut S has been found, we have

Proof of $O(\sqrt{\log n})$ guarantee

Note that,

$$\sum_{i,j \in V : i \neq j} \|v_i - v_j\|^2 \geq \sum_{i \in L, j \in R} \|v_i - v_j\|^2 \geq \Omega(n^2/\sqrt{\log n}).$$

Then,

$$\rho(S) \leq O(\sqrt{\log n}) \frac{1}{n^2} \sum_{e=(i,j) \in E} \|v_i - v_j\|^2 \leq O(\sqrt{\log n}).OPT.$$

The last inequality follows from the SDP relaxation.

Proof sketch of $O(\sqrt{\log n})$ guarantee

From theorem 17, once such a cut S has been found, we have

Proof of $O(\sqrt{\log n})$ guarantee

Note that,

$$\sum_{i,j \in V : i \neq j} \|v_i - v_j\|^2 \geq \sum_{i \in L, j \in R} \|v_i - v_j\|^2 \geq \Omega(n^2/\sqrt{\log n}).$$

Then,

$$\rho(S) \leq O(\sqrt{\log n}) \frac{1}{n^2} \sum_{e=(i,j) \in E} \|v_i - v_j\|^2 \leq O(\sqrt{\log n}).OPT.$$

The last inequality follows from the SDP relaxation.

Proof sketch of $O(\sqrt{log\ n})$ guarantee

From theorem 17, once such a cut S has been found, we have

Proof of $O(\sqrt{log\ n})$ guarantee

Note that,

$$\sum_{i,j \in V: i \neq j} \|v_i - v_j\|^2 \geq \sum_{i \in L, j \in R} \|v_i - v_j\|^2 \geq \Omega(n^2/\sqrt{log\ n}).$$

Then,

$$\rho(S) \leq O(\sqrt{log\ n}) \frac{1}{n^2} \sum_{e=(i,j) \in E} \|v_i - v_j\|^2 \leq O(\sqrt{log\ n}).OPT.$$

The last inequality follows from the SDP relaxation.

Experimental Setup

- MATLAB was used to implement all 3 algorithms.
- CVX, a package for specifying and solving convex programs was used in the implementation of Leighton-Rao and ARV algorithms.
- A total of 54 data sets were used from a wide variety of networks including biological, infrastructure, transportation, social,ecological, web, dynamic and brain networks.
- All data sets were obtained from http://networkrepository.com/
- We restricted the size of the graphs to about 60 nodes so that every algorithm could terminate within 15 minutes with a feasible solution.
- All experiements were run on a standard Intel Core i5 Proces-sor with dual processing capability (@3.2 GHz) and 4 GB of RAM.

Our results

- Let $\text{usc}_{SP}, \text{usc}_{LR}$ and usc_{ARV} denote the value of the uniform sparsest cut returned by the algorithms SP, LR and ARV.
- $O(\sqrt{log\ n})$ is still small for our problem instances and value of the constants are not accounted for, hence performance of ARV not much different from LR.
- ARV performed better on only some instances while on most others, it either output the same value(of USC) as LR or slightly higher.
- SP was the fastest-taking about 10-40 seconds. For modest sized inputs LR and ARV took 1-2 minutes, but for largest ones, about 10-15 minutes.

Our results

- Let usc_{SP}, usc_{LR} and usc_{ARV} denote the value of the uniform sparsest cut returned by the algorithms SP, LR and ARV.
- $O(\sqrt{log\ n})$ is still small for our problem instances and value of the constants are not accounted for, hence performance of ARV not much different from LR.
- ARV performed better on only some instances while on most others, it either output the same value(of USC) as LR or slightly higher.
- SP was the fastest-taking about 10-40 seconds. For modest sized inputs LR and ARV took 1-2 minutes, but for largest ones, about 10-15 minutes.

Our results

- Let usc_{SP}, usc_{LR} and usc_{ARV} denote the value of the uniform sparsest cut returned by the algorithms SP, LR and ARV.
- $O(\sqrt{log\ n})$ is still small for our problem instances and value of the constants are not accounted for, hence performance of ARV not much different from LR.
- ARV performed better on only some instances while on most others, it either output the same value(of USC) as LR or slightly higher.
- SP was the fastest-taking about 10-40 seconds. For modest sized inputs LR and ARV took 1-2 minutes, but for largest ones, about 10-15 minutes.

Our results

- Let usc_{SP}, usc_{LR} and usc_{ARV} denote the value of the uniform sparsest cut returned by the algorithms SP, LR and ARV.
- $O(\sqrt{log\ n})$ is still small for our problem instances and value of the constants are not accounted for, hence performance of ARV not much different from LR.
- ARV performed better on only some instances while on most others, it either output the same value(of USC) as LR or slightly higher.
- SP was the fastest-taking about 10-40 seconds. For modest sized inputs LR and ARV took 1-2 minutes, but for largest ones, about 10-15 minutes.

Our results

- The error between all 3 algorithms is small $\implies$ for most real world instances, all algorithms perform satisfactorily.
- SP is useful where the graph has expansion bounded by a constant, i.e $\Phi(G) = \Theta(1)$. For most real world data sets, we observe this is the case.
- On most instances, $\text{usc}_{LR} \leq \min\{\text{usc}_{SP}, \text{usc}_{ARV}\}$.

Our results

- $\text{usc}_{ARV} \leq \text{usc}_{SP}$ for more than 60% for the instances. SP does better than LR on very few inputs while ARV is worse than LR on roughly 60% of the instances.
- The average differences are as follows:

$$\frac{\text{usc}_{SP} - \text{usc}_{LR}}{\#ofinstances} = 0.0136 \; ; \; \frac{\text{usc}_{ARV} - \text{usc}_{LR}}{\#ofinstances} = 0.0187;$$

$$\frac{\text{usc}_{ARV} - \text{usc}_{SP}}{\#ofinstances} = 0.0051.$$

Plot of Overall performance

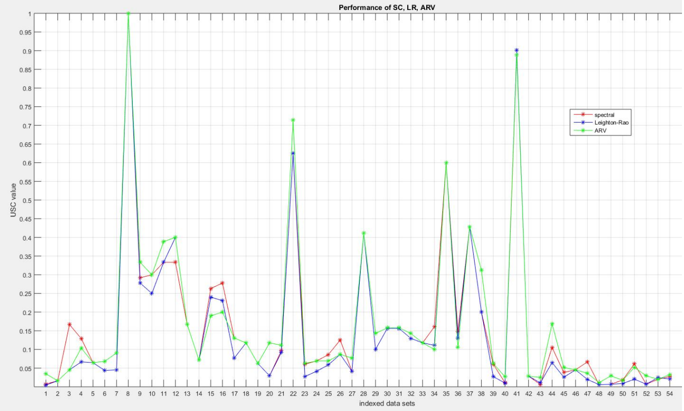

Figure 4.2: Note in particular how each of the 3 algorithms does not perform much worse than the rest.

Plot of SP vs LR

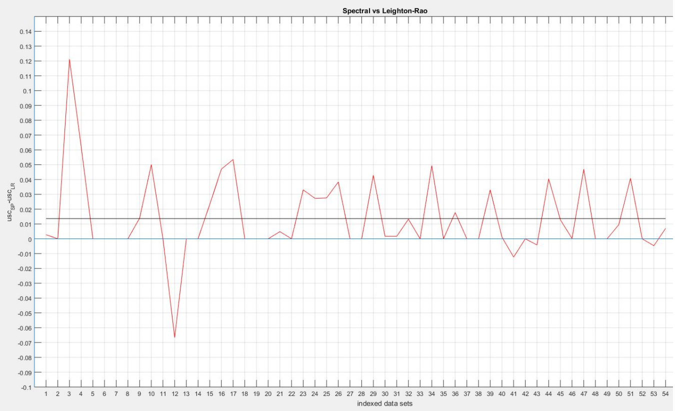

Figure 4.3: SP does strictly better than LR for exactly 4 instances. The average difference is small while the maximum difference is bounded by roughly 0.12.

Plot of ARV vs LR

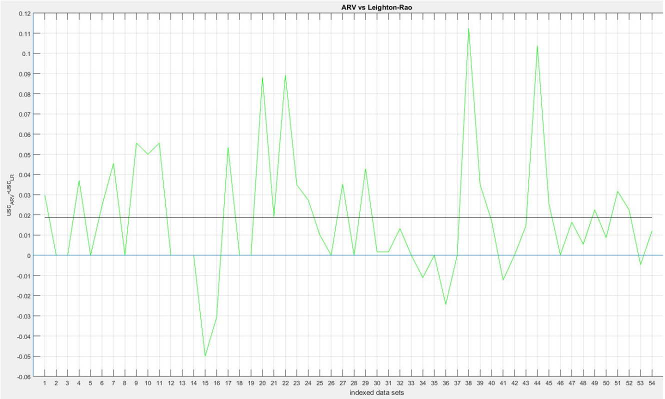

Figure 4.4: LR clearly does better than ARV in most instances. This is also reflected by the greater average difference than in Figure 6.4. The maximum difference is bounded by about 0.12

Plot of ARV vs SP

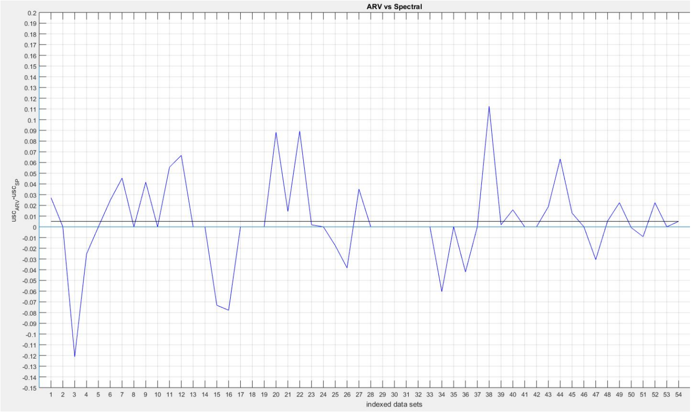

Figure 4.5: ARV does better than SP on most instances. However, the average difference is positive but small.

Takeaways

- Difficult to say which algorithm is truly the best among all.
- If running time is the concern, then SP quite well and outputs a cut of value which is close to both the LR and the ARV algorithm.
- However, if one wants the best cut and is willing to settle for a trade-off on running time, then either LR or ARV should be good choices.
- As n increases, both LR and ARV would take quite a significant amount of time with current implementation.
- Hence more efficient implementations need to be developed.

Future directions

- Research the usefulness of the cut-matching game and the multiplicative weights algorithm for solving USC problem as presented in AHK(2012).
- Research the usefulness of single-commodity framework for the USC problem presented in KRV(2009), and Orecchia et al (2008).
- Improve constants in the algorithms/reduce width of certain constraints, etc.

Thank you!

Q/A session.

References

Arora, Sanjeev, Satish Rao, and Umesh V. Vazirani (2004). "Expander flows, geometric embeddings and graph partitioning". In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*. Ed. by László Babai. ACM, pp. 222–231. ISBN: 1-58113-852-0. DOI: 10.1145/1007352.1007355. URL: http://doi.acm.org/10.1145/1007352.1007355.

Bourse, Florian, Marc Lelarge, and Milan Vojnovic (2014). "Balanced Graph Edge Partition". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA: ACM, pp. 1456–1465. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623660. URL: http://doi.acm.org/10.1145/2623330.2623660.

Călinescu, Gruia, Howard Karloff, and Yuval Rabani (1998). "An Improved Approximation Algorithm for Multiway Cut". In: *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*. STOC '98. Dallas, Texas, USA: ACM, pp. 48–52. ISBN: 0-89791-962-9. DOI: 10.1145/276698.276711. URL: http://doi.acm.org/10.1145/276698.276711.

Charikar, Moses and Vaggos Chatziafratis (2017). "Approximate Hierarchical Clustering via Sparsest Cut and Spreading Metrics". In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*. Ed. by Philip N. Klein. SIAM, pp. 841–854. ISBN: 978-1-61197-478-2. DOI: 10.1137/1.9781611974782.53. URL: https://doi.org/10.1137/1.9781611974782.53.

Charikar, Moses, Konstantin Makarychev, and Yury Makarychev (2006). "Near-optimal Algorithms for Unique Games". In: *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*. STOC '06. Seattle, WA, USA: ACM, pp. 205–214. ISBN: 1-59593-134-1. DOI: 10.1145/1132516.1132547. URL: http://doi.acm.org/10.1145/1132516.1132547.

Goemans, Michel X. and David P. Williamson (1995). "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming". In: *J. ACM* 42.6, pp. 1115–1145. ISSN: 0004-5411. DOI: 10.1145/227683.227684. URL: http://doi.acm.org/10.1145/227683.227684.

Nesterov, Yu (1998). "Semidefinite relaxation and nonconvex quadratic optimization". In: *Optimization Methods and Software* 9.1-3, pp. 141–160. DOI: 10.1080/10556789808805690. eprint: https://doi.org/10.1080/10556789808805690. URL: https://doi.org/10.1080/10556789808805690.

Swamy, Chaitanya (2004). "Correlation Clustering: Maximizing Agreements via Semidefinite Programming". In: *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '04. New Orleans, Louisiana: Society for Industrial and Applied Mathematics, pp. 526–527. ISBN: 0-89871-558-X. URL: http://dl.acm.org/citation.cfm?id=982792.982866.

Trevisan, Luca (2005). "Approximation Algorithms for Unique Games". In: *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*. FOCS '05. Washington, DC, USA: IEEE Computer Society, pp. 197–205. ISBN: 0-7695-2468-0. DOI: 10.1109/SFCS.2005.22. URL: https://doi.org/10.1109/SFCS.2005.22.

Wigderson, Avi (1983). "Improving the Performance Guarantee for Approximate Graph Coloring". In: *J. ACM* 30.4, pp. 729–735. ISSN: 0004-5411. DOI: 10.1145/2157.2158. URL: http://doi.acm.org/10.1145/2157.2158.