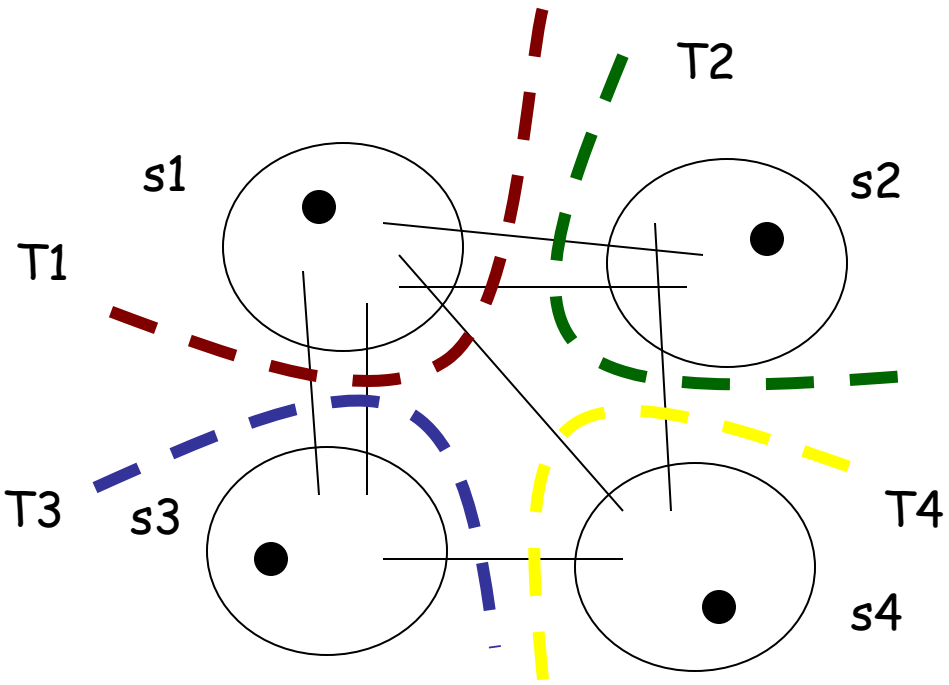
Approximation Algoirthms:
Graph Partitioning Problems



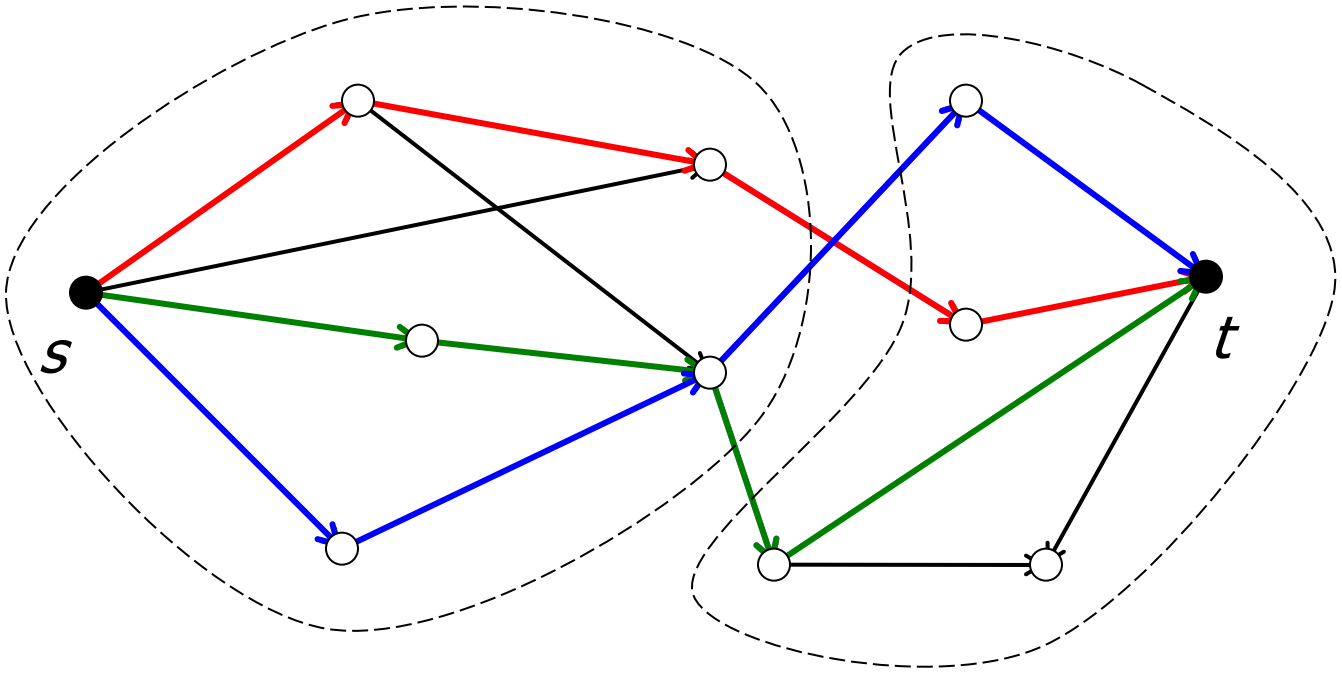
Graph Partitioning Problems

General setting: to remove a minimum (weight) set of edges to cut the graph into pieces.

Examples:

- Minimum (s-t) cut
- Multiway cut
- Multicut
- Sparsest cut
- Minimum bisection

Minimum s-t Cut



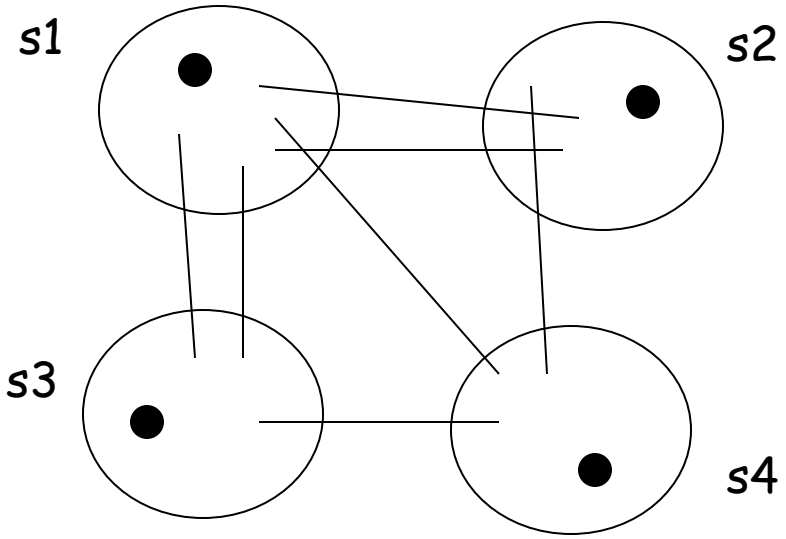
Minimum s-t cut = minimum (weighted) set of edges to disconnect s and t

Mininium s-t cut = Max s-t flow

Multiway Cut

Given a set of terminals S = {s1, s2, …, sk},
a **multiway cut** is a set of edges whose removal disconnects the terminals from each other.

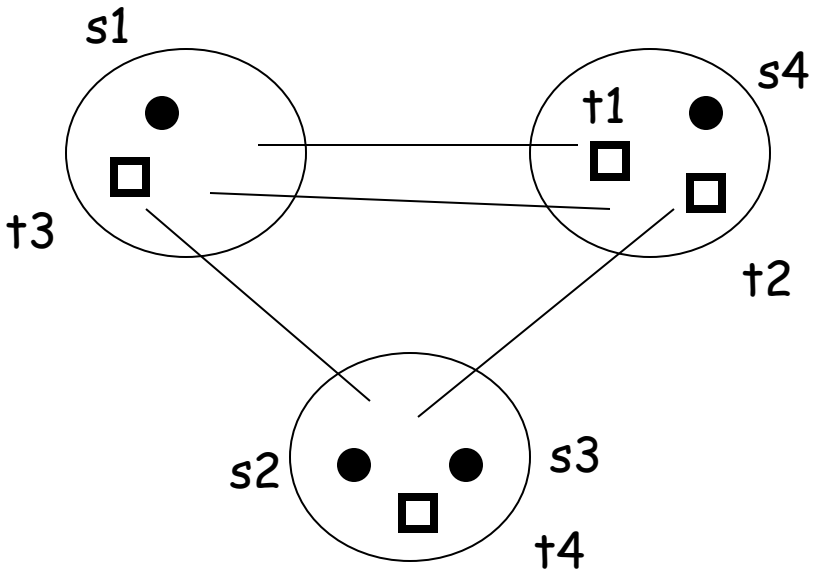
The **multiway cut problem** asks for the minimum weight multiway cut.



Multicut

Given k source-sink pairs {(s1,t1), (s2,t2), ...,(sk,tk)},
a **multicut** is a set of edges whose removal disconnects each source-sink pair.

The **multicut problem** asks for the minimum weight multicut.



Multicut vs Multiway cut

Given a set of terminals S = {s1, s2, …, sk},
a **multiway cut** is a set of edges whose removal disconnects the terminals from each other.

Given k source-sink pairs {(s1,t1), (s2,t2), …,(sk,tk)},
a **multicut** is a set of edges whose removal disconnects each source-sink pair.

What is the relationship between these two problems?

Multicut is a generalization of multiway cut. Why?

Because we can set each (si,sj) as a source-sink pair.

Sparsest Cut

Given k source-sink pairs {(s1,t1), (s2,t2), ...,(sk,tk)}.

For a set of edges U, let c(U) denote the total weight.
Let dem(U) denote the number of pairs that U disconnects.

The **sparsest cut problem** asks for a set U which minimizes c(U)/dem(U).

In other words, the sparsest cut problem asks for the most cost effective way to disconnect source-sink pairs, i.e. the average cost to disconnect a pair is minimized.

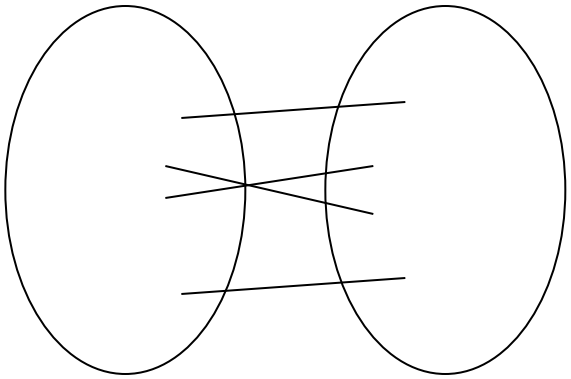
Sparsest Cut

Suppose every pair is a source-sink pair.

For a set of edges U, let c(U) denote the total weight.
Let dem(U) denote the number of pairs that U disconnects.

The **sparsest cut problem** asks for a set U which minimizes c(U)/dem(U).

S

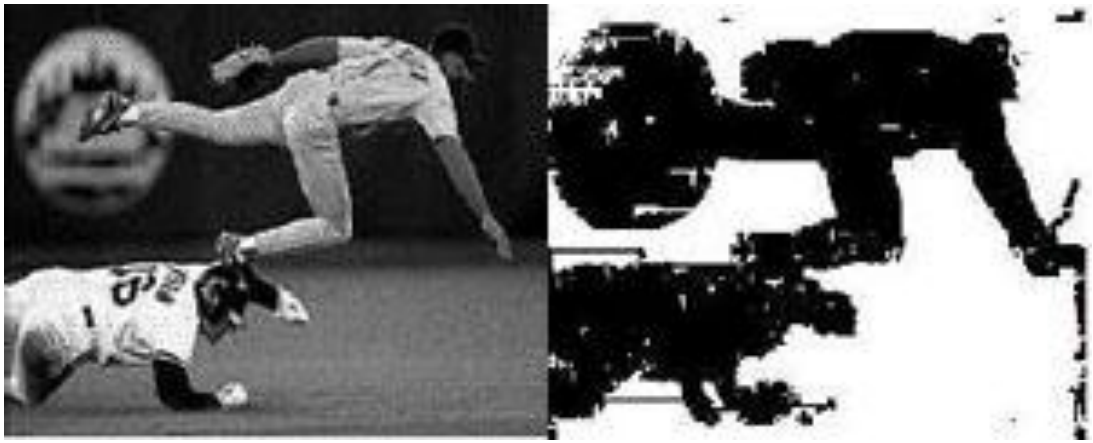


V-S

Minimize $\dfrac{c(\delta(S))}{|S| \cdot |V - S|}$

Sparsest Cut

This is related to the normalized cut in image segmentation.

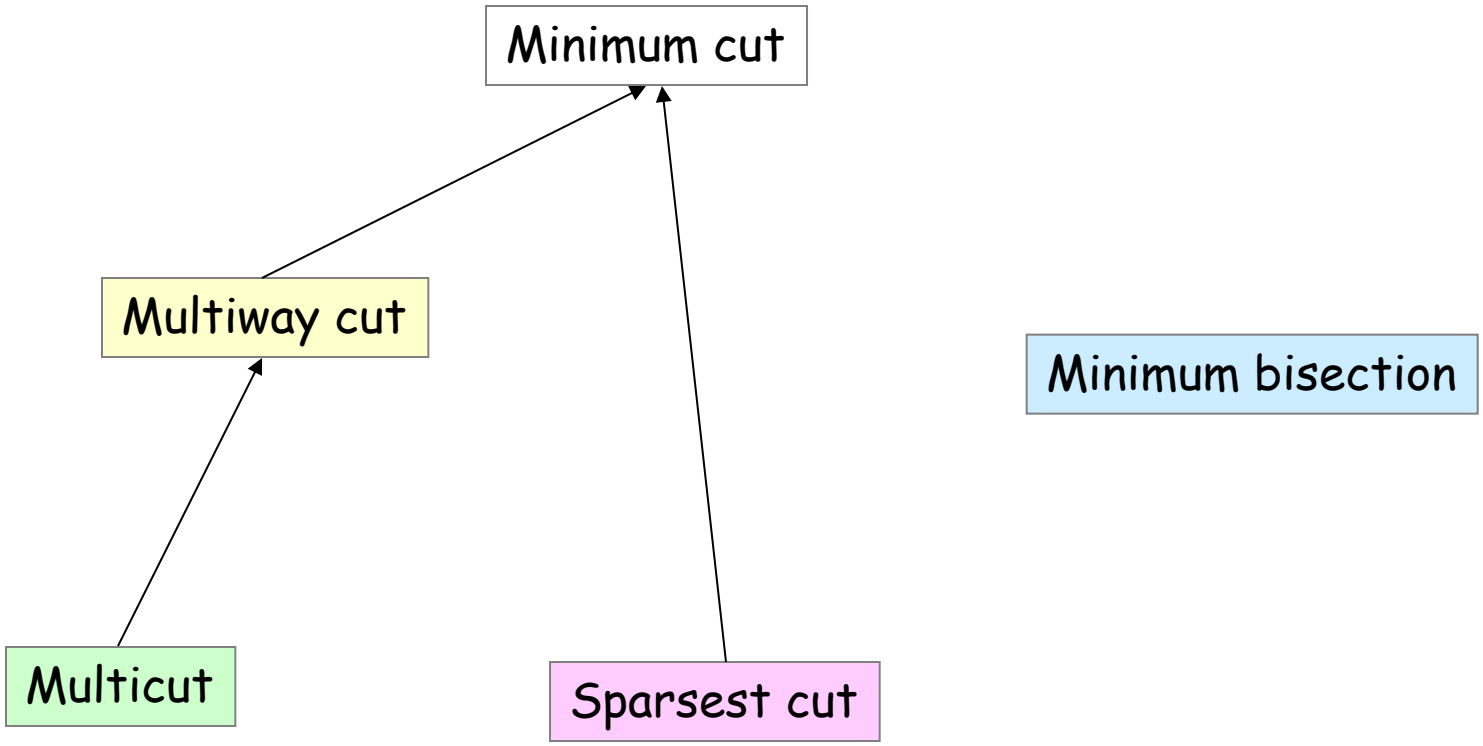


Minimum Bisection

The **minimum bisection problem** is to divide the vertex set into two equal size parts and minimize the total weights of the edges in between.

This problem is very useful in designing approximation algorithms for other problems – to use it in a divide-and-conquer strategy.

Relations


Minimum cut
Multiway cut
Minimum bisection
Multicut
Sparsest cut

Results

Minimum cut

- Polynomial time solvable.

Multiway cut

- a combintorial 2-approximation algorithm
- an elegant LP-based 1.34-approximation

Multicut

- an $O(\log n)$-approximation algorithm.
- some "evidence" that no constant factor algorithm exists.

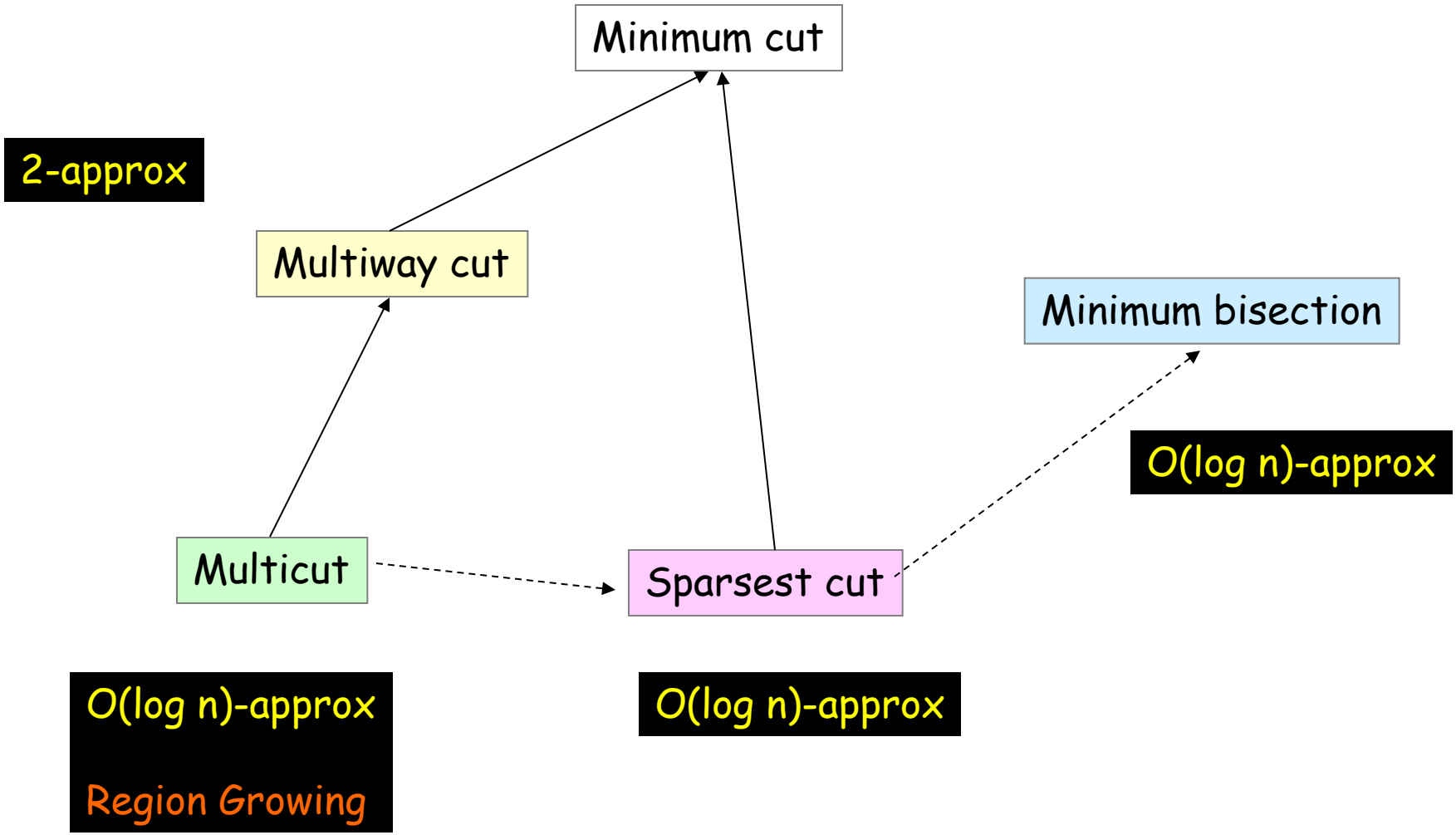
Sparsest cut

- an $O(\log n)$-approximation algorithm based on multicut.
- an $O(\sqrt{\log n})$-approximation based on semidefinite programming.
- some "evidence" that no constant factor algorithm exists.

Min bisection

- an $O(\log n)$-approximation algorithm based on sparsest cut. (this statement is not quite accurate but close enough).

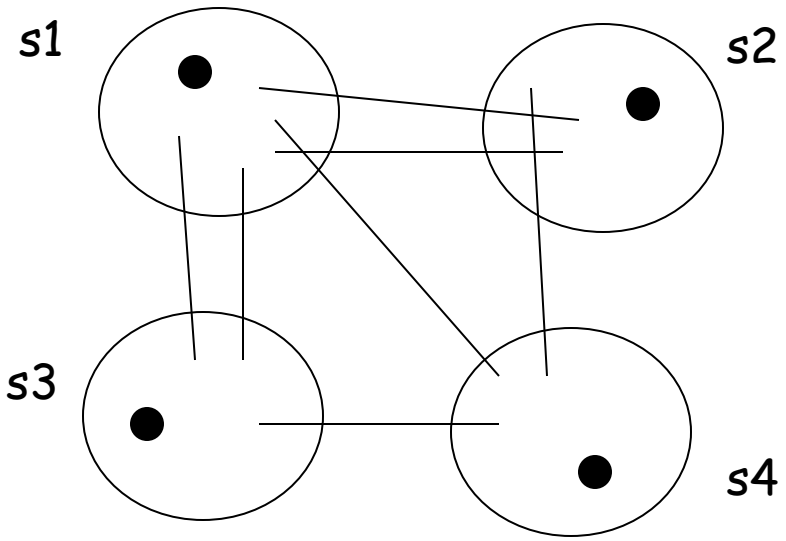
Relations


Minimum cut
2-approx
Multiway cut
Minimum bisection
O(log n)-approx
Multicut
Sparsest cut
O(log n)-approx
O(log n)-approx
Region Growing

Multiway Cut

Given a set of terminals S = {s1, s2, …, sk},
a **multiway cut** is a set of edges whose removal disconnects the terminals from each other.

The **multiway cut problem** asks for the minimum weight multiway cut.



This picture leads to a natural algorithm!

Algorithm

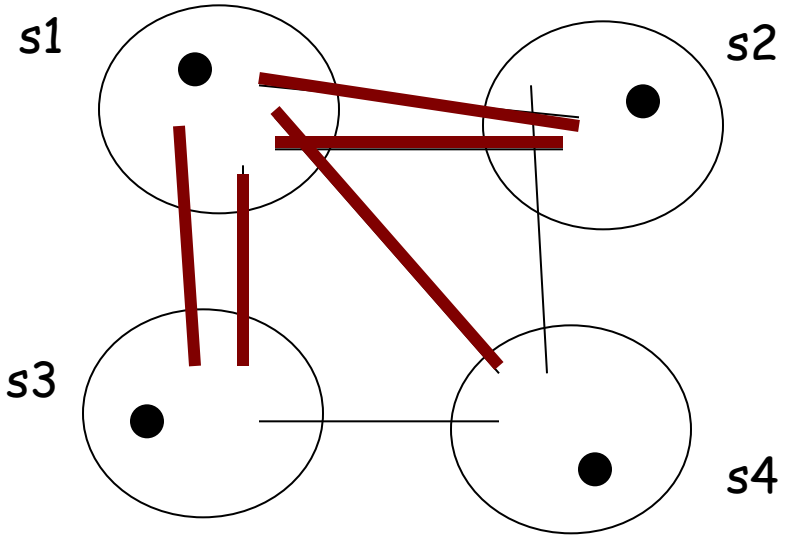
Define an **isolating cut** for s(i) to be a set of edges whose removal disconnects s(i) from the rest of the terminals.

(Multiway cut 2-approximation algorithm)

1. For each i, compute a minimum weight isolating cut for s(i), say $C(i)$.
2. Output the union of $C(i)$.

How to compute a minimum isolating cut?

Analysis

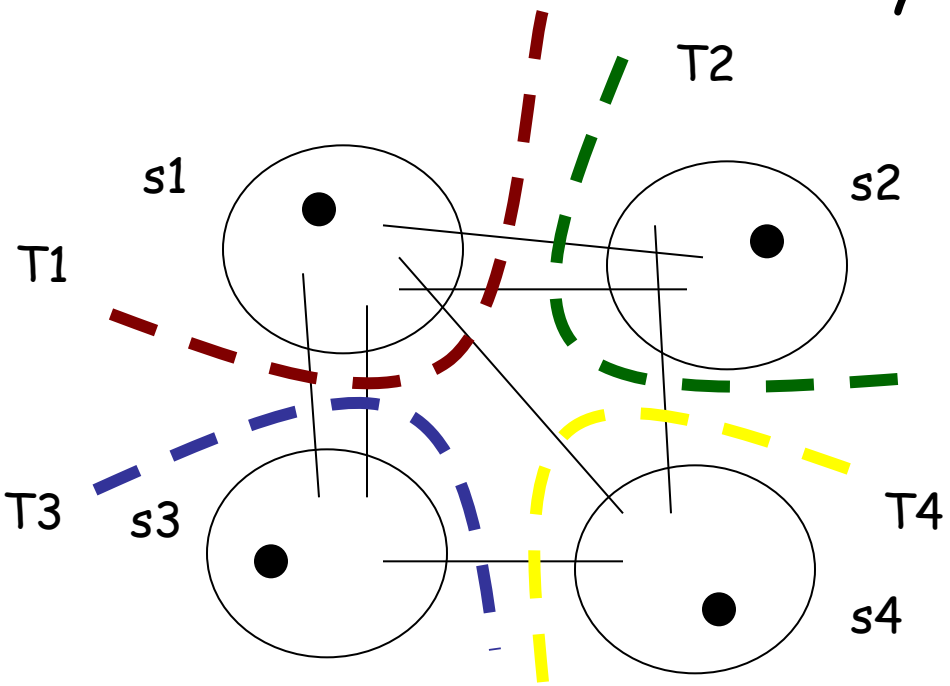


Why is it a 2-approximation?

Imagine this is an optimal solution.

The (thick) red edges form an isolating cut for s1, call it T1.
Since we find a minimum isolating cut for s1, we have w(C1) <= w(T1).

Analysis



Why is it a 2-approximation?

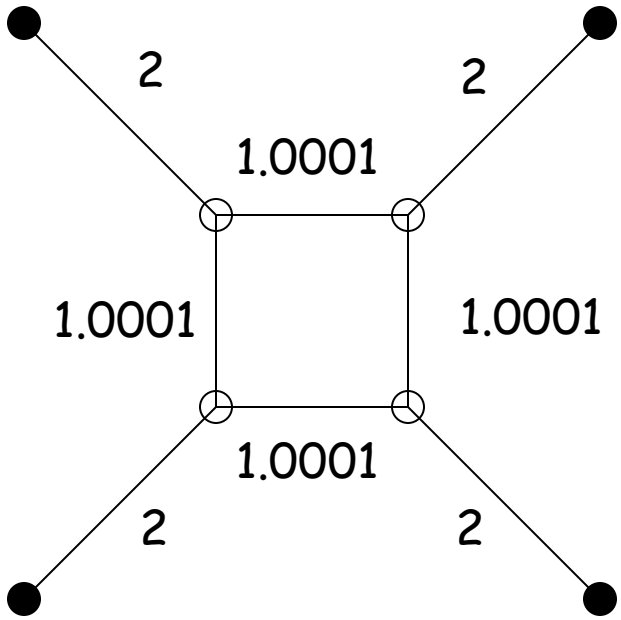
Imagine this is an optimal solution.

Key: w(Ci) <= w(Ti)

- ALG = w(C1) + w(C2) + w(C3) + w(C4)
- OPT = (w(T1) + w(T2) + w(T3) + w(T4)) / 2

So, ALG <= 2OPT.

Bad Example

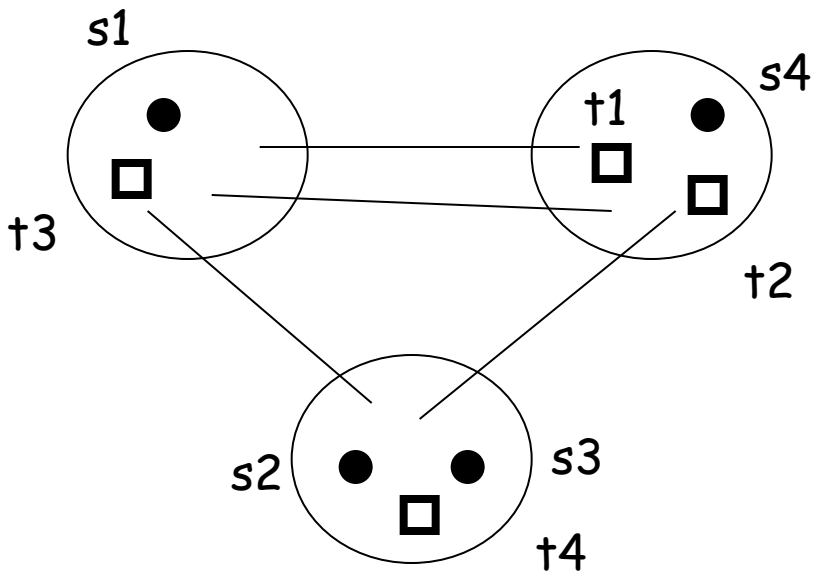


- OPT = 4.0004
- ALG = 8

Multicut

Given k source-sink pairs {(s1,t1), (s2,t2), …,(sk,tk)},
a **multicut** is a set of edges whose removal disconnects each source-sink pair.

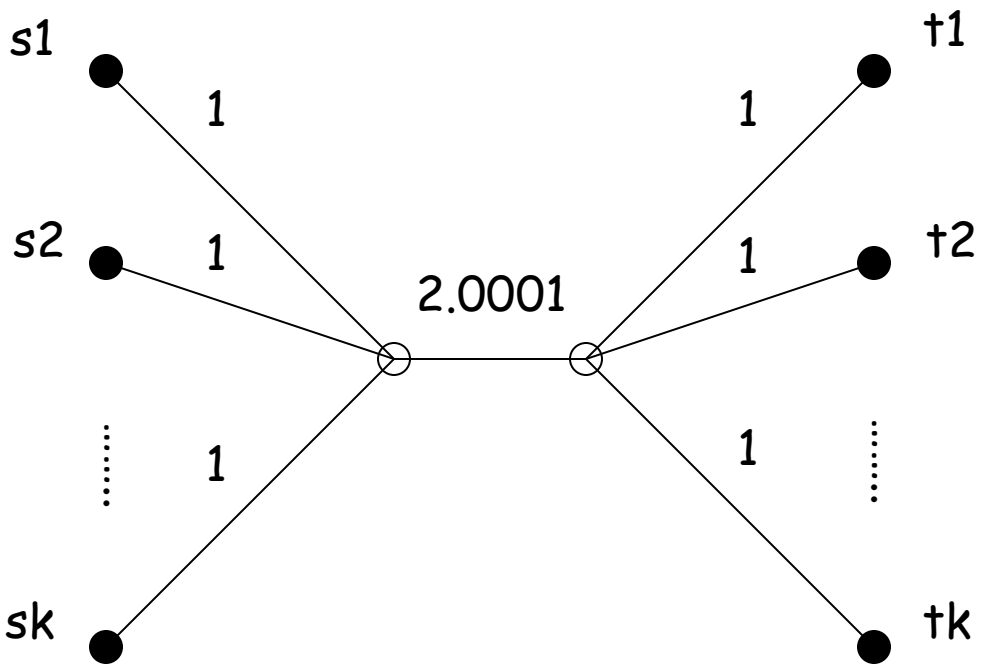
The **multicut problem** asks for the minimum weight multicut.



Can we use the idea in the isolating cut algorithm?

Bad Example

Algorithm: take the union of minimum si-ti cut.



- OPT = 2.0001
- ALG = 2k

Linear Program

$$\min \sum_{e \in E(G)} c_e d_e$$

$$\sum_{e \in p} d_e \geq 1$$ for each path p connecting a source-sink pair

$$d_e \geq 0$$

Separation oracle: given a fractional solution d, decide if d is feasible.

Shortest path computations between source-sink pairs.

Rounding

$$\min \sum_{e \in E(G)} c_e d_e$$

$$\sum_{e \in p} d_e \geq 1$$ for each path p connecting a source-sink pair

$$d_e \geq 0$$

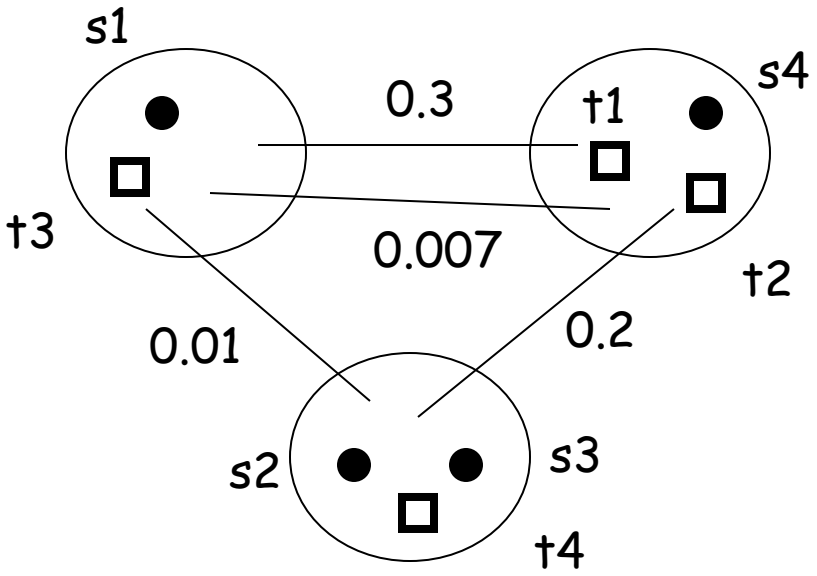
Intuitively, we would like to take edges with large d(e).

Fractional solution could be very fractional.

Look at a vertex solution.

Don't know what to say…

Strategy



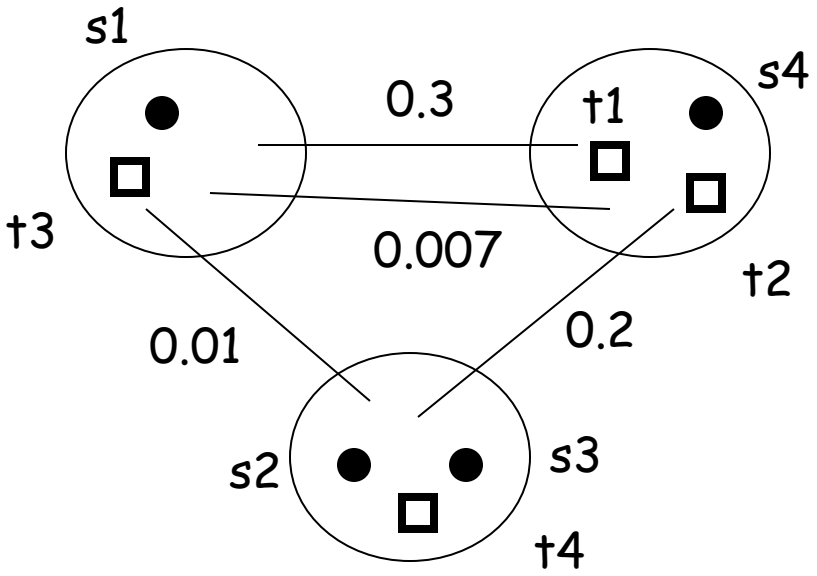
Let the edges in this multicut be C.

Given the fractional value of d(e),
how can we compare a multicut with the optimal value of the LP?

It would be good if $d(e) \geq 1/2$ (or $1/k$) for every edge in C.
Then we would have a 2-approximation algorithm (or k-approximation algorithm).

But this is not true.

Strategy



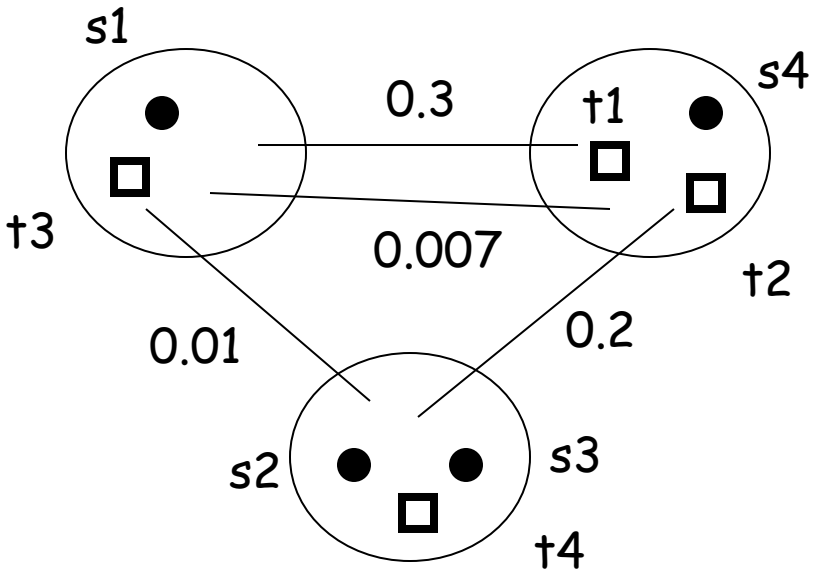
Let the edges in this multicut be C.

Given the fractional value of d(e),
how can we compare a multicut with the optimal value of the LP?

It would also be good if $\sum c(e) \leq k \sum c(e)d(e)$ for edges in C.
Then we would have a k-approximation algorithm.

But this is also not true.

Strategy



Let the edges in this multicut be C.

Observation:
we haven't considered the edges inside the components.

Analysis strategy:

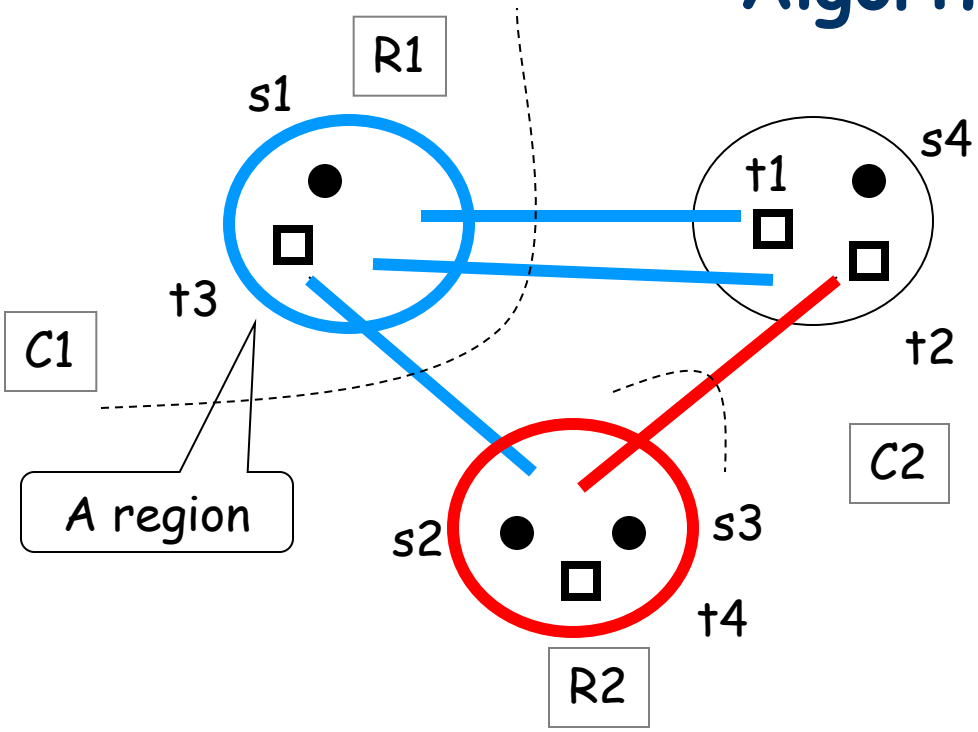
If we can prove that $\sum_{e \in C} c(e) \leq f(n)(\sum_{e \in E(G)} c(e)d(e)) = f(n) \cdot OPT$

then we have a f(n)-approximation algorithm.

We'll use this strategy.

How to find such a multicut C?

Algorithm



Goal: Find a cut with

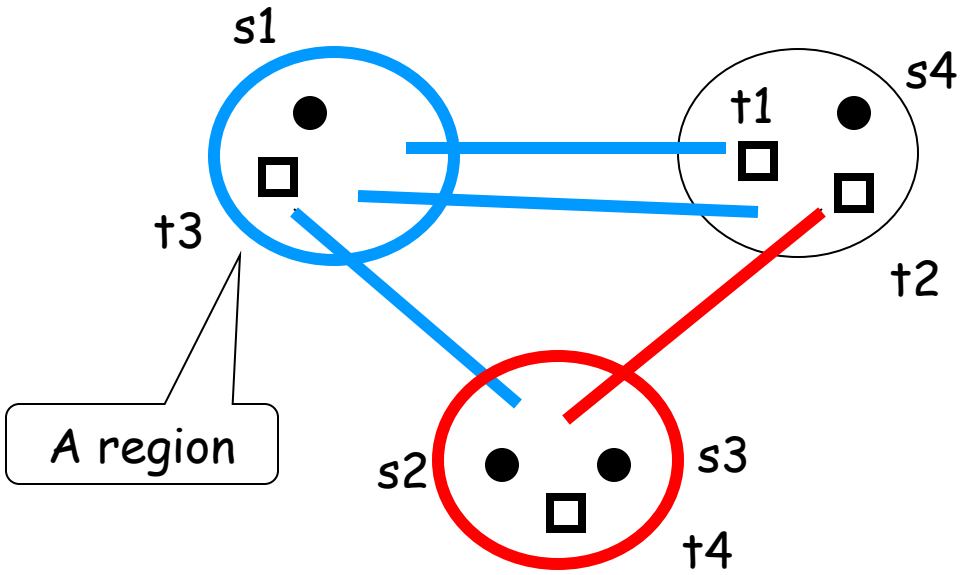
$$\sum_{e \in C} c(e) \leq f(n) \sum_{e \in E(G)} c(e)d(e)$$

(Multicut approximation algorithm)

1. For each i, **compute** a s(i)-t(i) cut, say C(i).
2. **Remove** C(i) and its component R(i) (its region) from the graph

Output the union of C(i).

Requirements



Goal: Find a cut with

$$\sum_{e \in C} c(e) \leq f(n) \sum_{e \in E(G)} c(e)d(e)$$

What do we need for C(i)?

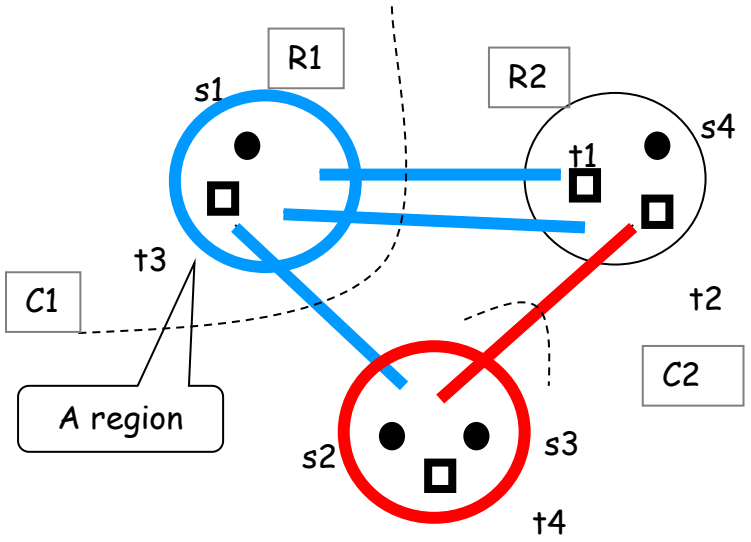
Cost requirement:

$$\sum_{e \in C(i)} c(e) \leq f(n) \cdot \sum_{e \in C(i) \cup R(i)} c(e) \cdot d(e)$$

Feasibility requirement:

There is no source-sink pair in each R(i).

Cost Requirement



Goal: Find a cut with

$$\sum_{e \in C} c(e) \leq f(n) \sum_{e \in E(G)} c(e)d(e)$$

Cost requirement:

$$\sum_{e \in C(i)} c(e) \leq f(n) \cdot \sum_{e \in C(i) \cup R(i)} c(e) \cdot d(e)$$

Cost requirement implies the **Goal:**

$$\sum_{e \in C} c(e) = \sum_{i} \sum_{e \in C(i)} c(e) \leq f(n) \cdot \sum_{i} \sum_{e \in C(i) \cup R(i)} c(e) \cdot d(e) \leq f(n) \sum_{e \in E(G)} c(e) \cdot d(e)$$

It is important that every edge is counted at most once,
and this is why we need to remove C(i) and R(i) from the graph.

Linear Program

Question: How to find the cut, i.e. R(i) and C(i), to satisfy the requirements?

$$\min \quad \sum_{e \in E(G)} c_e d_e$$

$$\sum_{e \in p} d_e \geq 1 \qquad \text{for each path p connecting a source-sink pair}$$

$$d_e \geq 0$$

A useful interpretation is to think of d(e) as the length of e.

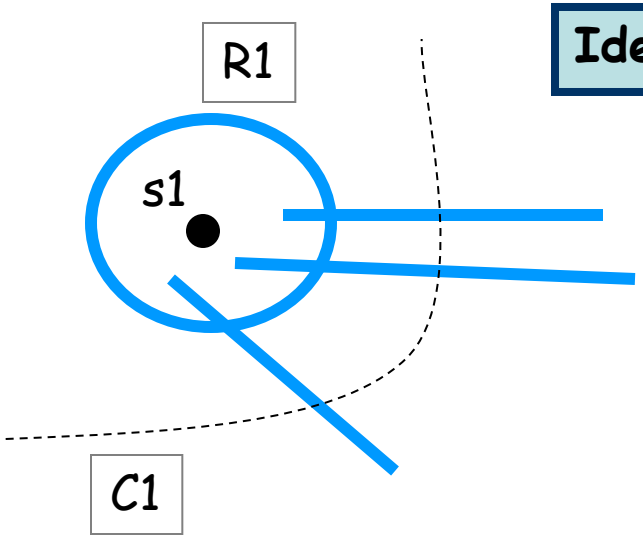
So the linear program says that each source-sink pair is of distance at least 1.

Distance

Key: think of d(e) as the length of e.

Define the distance between two vertices as the length of their shortest path.

Given a vertex v as the center,
define S(r) to be the set of vertices of distance at most r from v.



Idea: Set R(i) be to be S(r) with s1 as the center.

Then, naturally, set C(i) to be the set of edges with one endpoint in R(i) and one endpoint outside R(i).

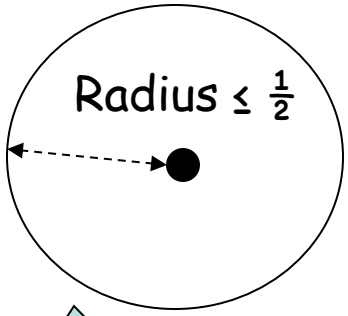
Feasibility Requirement

Feasibility requirement:

There is no source-sink pair in each R(i).

This is because we'll remove R(i) from the graph.

The linear program says that each source-sink pair is of distance at least 1.



A region defined by a ball

Idea: Only choose S(r) with $r \le \frac{1}{2}$.

Since the distance between s(i) and t(i) is at least 1,
they cannot be in the same R(j),
and hence the feasibility requirement is satisfied.

Where are we?

(Multicut approximation algorithm)

1. For each i, **compute** a s(i)-t(i) cut, say C(i).
2. **Remove** C(i) and its component R(i) (its region) from the graph

Output the union of C(i).

Use the idea of ball to find R(i) and C(i)

The ball has to satisfy two requirements:

Cost requirement:

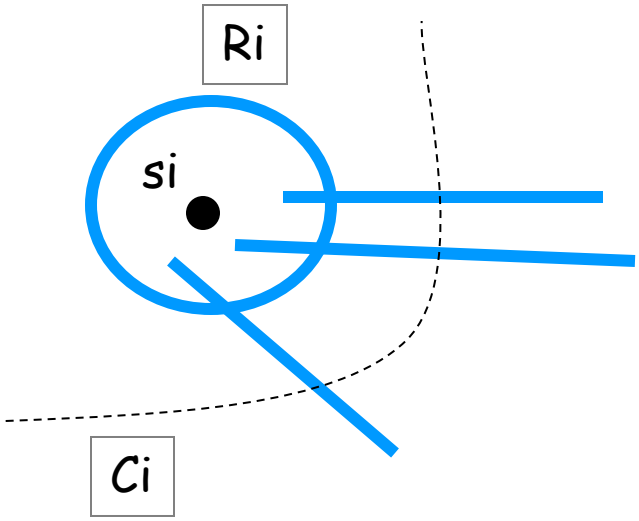
$$\sum_{e \in C(i)} c(e) \leq f(n) \cdot \sum_{e \in C(i) \cup R(i)} c(e) \cdot d(e)$$

Feasibility requirement:

There is no source-sink pair in each R(i).

By choosing the radius at most $\frac{1}{2}$

Finding Cheap Regions



Cost requirement:

$$\sum_{e \in C(i)} c(e) \leq f(n) \cdot \sum_{e \in C(i) \cup R(i)} c(e) \cdot d(e)$$

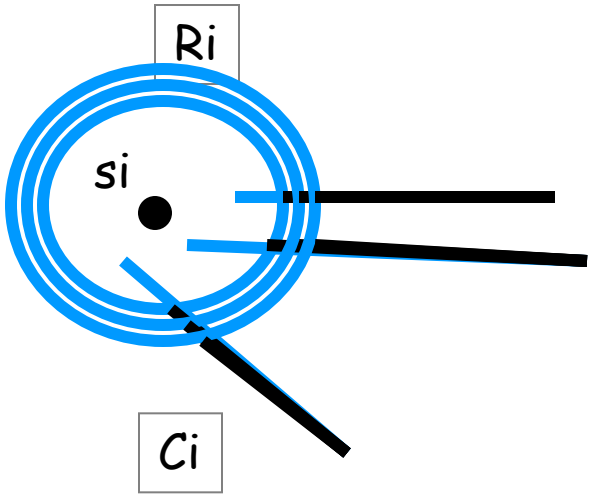
Want: f(n) to be as small as possible.

Region growing: search from S(0) to S(1/2)!

Continuous process: think of edges as infinitely short.

1. set R(i) = S(r); initially r=0.
2. check if cost requirement is satisfied.
3. if not, increase r and repeat.

Exponential Increase



Cost requirement:

$$\sum_{e \in C(i)} c(e) \leq f(n) \cdot \sum_{e \in C(i) \cup R(i)} c(e) \cdot d(e)$$

$c(S(r))$ $wt(S(r))$

If the cost requirement is not satisfied, we make the ball bigger.

Note that the right hand side increases in this process, and so the left hand side also increases faster, and so on.

In fact, the right hand side grows exponentially with the radius.

$$d\ wt(S(r)) \geq c(S(r))dr > f(n)wt(S(r))dr \Rightarrow wt(S(r)) = wt(S(0))e^{f(n)\cdot r}$$

Logarithmic Factor

$$wt(S(r)) = wt(S(0))e^{f(n)\cdot r}$$

Let $F = \min \sum_{e \in E(G)} c_e d_e$, the optimal value of the LP.

We only need to grow k regions, where k is the number of source-sink pairs.

Set wt(S(0)) = F/k.
In other words, we assign some additional weights to each source, but the total additional weight is at most F.

Maximum weight a ball can get is F + F/k, from all the edges and the source.

Set f(n) = 2ln(k+1).

$$wt(S(1/2)) = wt(S(0))e^{2\ln(k+1)\cdot\frac{1}{2}} \geq \frac{F}{k}\cdot e^{\ln(k+1)} \geq \frac{F}{k}\cdot k = F$$

Logarithmic Factor

To summarize:
By using the technique of region growing,
we can find a cut (a ball with radius at most $\frac{1}{2}$) that satisfies:

Cost requirement:

$$\sum_{e \in C(i)} c(e) \leq O(\ln k) \cdot \sum_{e \in C(i) \cup R(i)} c(e) \cdot d(e)$$

Feasibility requirement:

There is no source-sink pair in each R(i).

The cost requirement implies that it is an O(ln k)-approximation algorithm.

The analysis is tight. The integrality gap of this LP is acutally $\Omega(\ln k)$.

The Algorithm

(Multicut approximation algorithm)

Solve the linear program.

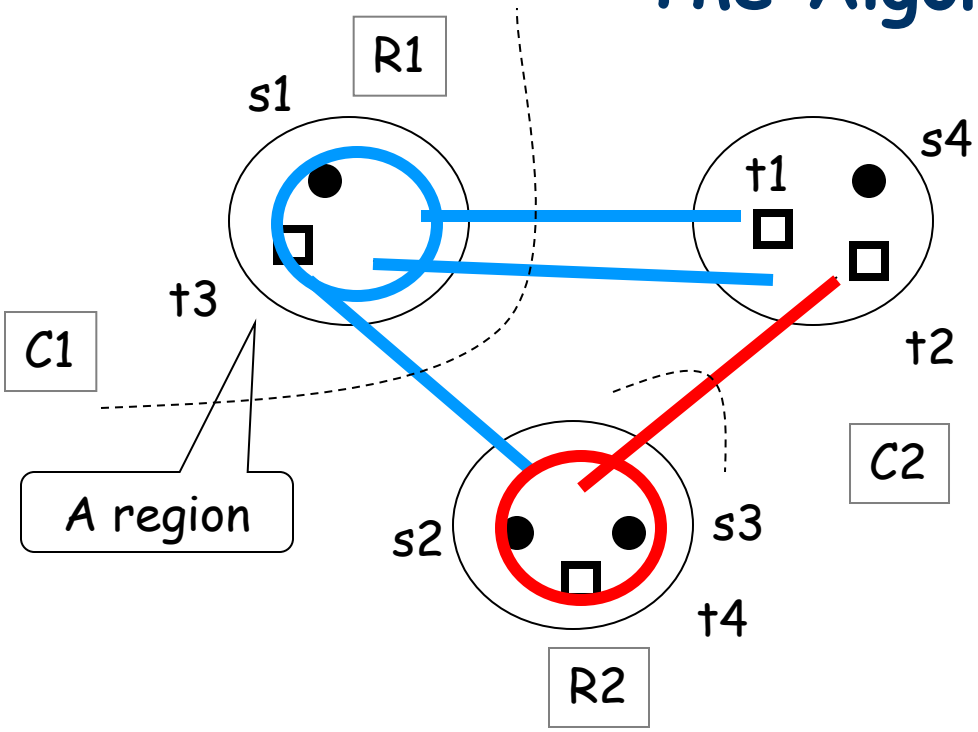
1. For each i, **compute** a s(i)-t(i) cut, say C(i).
2. **Remove** C(i) and its component R(i) (its region) from the graph

Output the union of C(i).

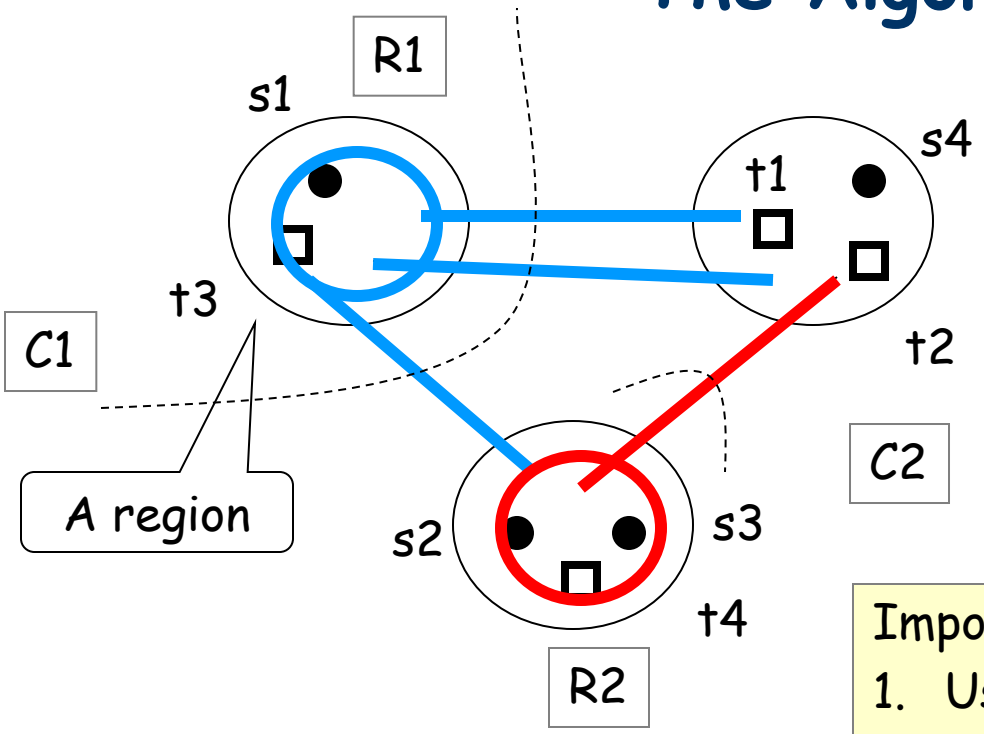
(Region growing algorithm)

1. Assign a weight F/k to s(i), and set S={s(i)}.
2. Add vertices to S in increasing order of their distances from s(i).
3. Stop at the first point when c(S), the total weight of the edges on the boundary, is at most 2ln(k+1)wt(S).
4. Set R(i):=S, and C(i) be the set of edges crossing R(i).

The Algorithm



The Algorithm



Important ideas:

1. Use linear program.
2. Compare the cost of the cut to the cost of the region.
3. Think of the variables as distances.
4. Growing the ball to find the region.

The idea of region growing can also be applied to other graph problems, most notably the feedback arc set problem, and also many applications.

Semidefinite Programming

What is semidefinite programming?

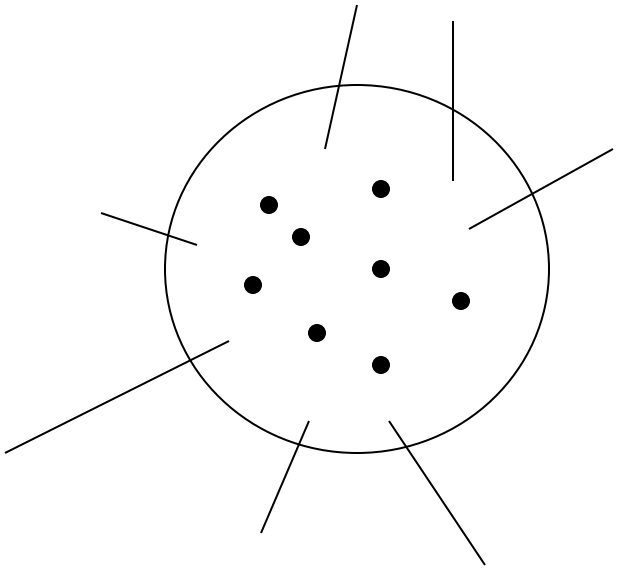
- A generalization of linear programming.
- A special case of convex programing.
- **Can be optimized in polynomial time.**

What is it good for?

- Shannon capacity
- Perfect graphs
- Number theory
- Quantum computation
- **Approximation algorithms**
  - Image segmentation and clustering
  - Constraint satisfaction problems
- etc…

Maximum Cut

(Maximum Cut)
Given an undirected graph, with an edge weight w(e) on each edge e, find a partition (S,V-S) of V so as to maximum the total weight of edges in this cut, i.e. edges that have one endpoint in S and one endpoint in V-S.



Maximum Cut

When is computing maximum cut easy?

When we are given a bipartite graph.

The maximum cut problem can also be interpreted as the problem of finding a **maximum bipartite subgraph**.

There is a simple greedy algorithm with approximation ratio $\frac{1}{2}$.

Similar to vertex cover.

Quadratic Program for MaxCut

$y_i = \{-1, +1\}$ The two sides of the partition.

$y_i \cdot y_j = -1$ if they are on opposite sides.

$y_i \cdot y_j = +1$ if they are on the same side.

$$\max \quad \frac{1}{2} \sum_{1 \le i \le j \le n} w_{ij}(1 - y_i \cdot y_j)$$

$$y_i^2 = 1$$

Quadratic Program for MaxCut

$$\max \quad \frac{1}{2} \sum_{1 \le i \le j \le n} w_{ij}(1 - y_i \cdot y_j)$$

$$y_i^2 = 1$$

This quadratic program is called **strict quadratic program,** because every term is of degree 0 or degree 2.

Of course this is unlikely to be solved in polynomial time, otherwise P=NP.

Vector Program for MaxCut

$$\max \quad \frac{1}{2} \sum_{1 \le i \le j \le n} w_{ij}(1 - y_i \cdot y_j)$$

$$y_i^2 = 1$$

$$y_i \in \mathbf{Z}$$

$$\max \quad \frac{1}{2} \sum_{1 \le i \le j \le n} w_{ij}(1 - v_i \cdot v_j)$$

$$v_i^2 = 1$$

$$v_i \in \mathbf{R}^n$$

Vector Program for MaxCut

$$\max \quad \frac{1}{2} \sum_{1 \le i \le j \le n} w_{ij}(1 - v_i \cdot v_j)$$

$$v_i^2 = 1$$

$$v_i \in \mathbf{R}^n$$

This is a relaxation of the strict quadratic program (why?)

Vector program: linear inequalities over inner products.

Vector program = semidefinite program.

Can be "solved" in polynomial time (ellipsoid, interior point).

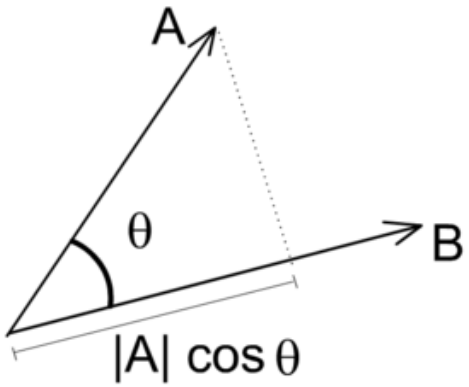
Geometric Interpretation

$$\max \quad \frac{1}{2} \sum_{1 \le i \le j \le n} w_{ij}(1 - v_i \cdot v_j)$$

$$v_i^2 = 1$$

$$v_i \in \mathbf{R}^n$$

Think of v_i as an n-dimensional vector.



$$v_i \cdot v_j = \cos \theta_{ij}$$

Contribute more to the objective if they are more far apart.

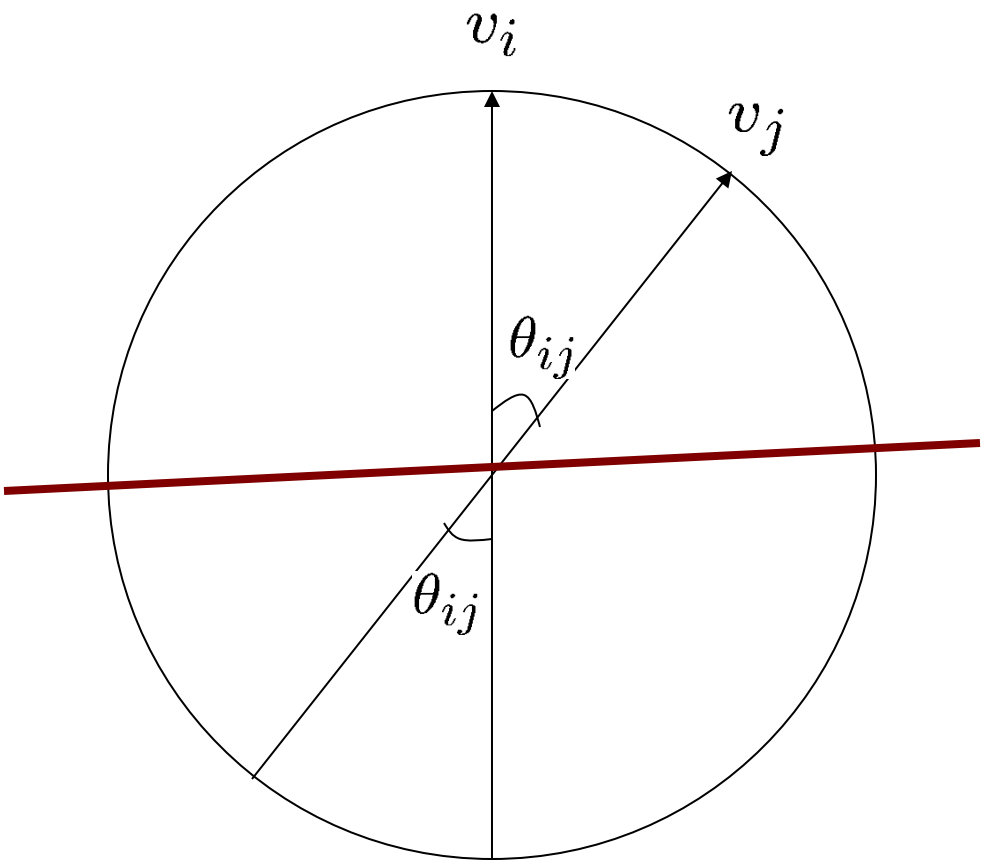
Algorithm

(Max-Cut Algorithm)

1. Solve the vector program. Let $a_1, \ldots, a_n$ be an optimal solution.
2. Pick r to be a uniformly distributed vector on the unit sphere S_{n-1}.
3. Let $S = \{v_i \mid a_i \cdot r \geq 0\}$

Analysis

Claim: $\mathbf{Pr}[v_i \text{ and } v_j \text{ are separated }] = \dfrac{\theta_{ij}}{\pi}$



Analysis

$$\max \quad \frac{1}{2} \sum_{1 \le i \le j \le n} w_{ij}(1 - v_i \cdot v_j)$$

Suppose v_i and v_j has an edge.

Contribution to semidefinite program: $\frac{w_{ij}}{2}(1 - \cos\theta_{ij})$

Contribution to the solution: $w_{ij} \cdot \mathbf{Pr}[v_i \text{ and } v_j \text{ are separated }] = w_{ij} \cdot \frac{\theta_{ij}}{\pi}$

Approximation Ratio: $\frac{2}{\pi} \cdot \frac{\theta_{ij}}{(1 - \cos\theta_{ij})} > 0.87856$

Analysis

Proof: Linearity of expection.

Let W be the random variable denoting the weight of edges in the cut.

Claim: $E(W) \geq 0.87856 \cdot OPT$

$$E(W) = \sum_{1 \leq i < j \leq n} w_{ij} \cdot \mathbf{Pr}[v_i \text{ and } v_j \text{ are separated}]$$

$$= \sum_{1 \leq i < j \leq n} w_{ij} \cdot \frac{\theta_{ij}}{\pi} \geq 0.87856 \cdot \sum_{1 \leq i < j \leq n} \frac{1}{2} w_{ij}(1 - \cos\theta_{ij})$$

$$\geq 0.87856 \cdot OPT$$

Algorithm

(Max-Cut Algorithm)

1. Solve the vector program. Let $a_1, \ldots, a_n$ be an optimal solution.
2. Pick r to be a uniformly distributed vector on the unit sphere S_{n-1}.
3. Let $S = \{v_i \mid a_i \cdot r \geq 0\}$

Repeat a few times to get a good approximation with high probability.

This algorithm performs extremely well in practice.

Try to find a tight example.

Remarks

Hard to imagine a combinatorial algorithm with the same performance.

Assuming the "unique games conjecture",
this algorithm is the best possible!

That is, it is NP-hard to find a better approximation algorithm!

Constraint Satisfaction Problems

(Max-2-SAT)

Given a formula in which each clause contains two literals,

find a truth assignment that satisfies the maximum number of clauses.

e.g. $(x_1 \vee x_2) \wedge (\overline{x_2} \vee x_5) \wedge \ldots \wedge (x_3 \vee x_1)$

An easy algorithm with approximation ratio $\frac{1}{2}$.

An LP-based algorithm with approximation ratio $\frac{3}{4}$.

An SDP-based algorithm with approximation ratio 0.87856.

Vector Program for MAX-2-SAT

(Max-2-SAT)
Given a formula in which each clause contains two literals,
find a truth assignment that satisfies the maximum number of clauses.

$$y_i = \{-1, +1\}$$

Additional variable (trick): $y_0 = \{-1, +1\}$

A variable is set to be true if: $y_i = y_0$

A variable is set to be false if: $y_i \neq y_0$

Vector Program for MAX-2-SAT

Denote v(C) to be the value of a clause C, which is defined as follows.

$$v(x_i) = \frac{1 + y_0 y_i}{2} \qquad v(\overline{x_i}) = \frac{1 - y_0 y_i}{2}$$

Consider a clause containing 2 literals, e.g. $(x_i \vee x_j)$. Its value is:

$$\begin{aligned} v(x_i \vee x_j) &= 1 - v(\overline{x_i}) v(\overline{x_j}) = 1 - \frac{1 - y_0 y_i}{2} \frac{1 - y_0 y_j}{2} \\ &= \frac{1}{4}(3 + y_0 y_i + y_0 y_j - y_0^2 y_i y_j) \\ &= \frac{1 + y_0 y_i}{4} + \frac{1 + y_0 y_j}{4} + \frac{1 - y_i y_j}{4} \end{aligned}$$

Vector Program for MAX-2-SAT

Objective: $\max \sum v(C)$

$$\max \sum_{0 \le i < j \le n} [a_{ij}(1 + v_i \cdot v_j) + b_{ij}(1 - v_i \cdot v_j)]$$

$$v_i^2 = 1$$

$$v_i \in \mathbf{R}^{n+1}$$

where a(ij) and b(ij) is the sum of coefficients.

Algorithm

(MAX-2-SAT Algorithm)

1. Solve the vector program. Let $a_1, \ldots, a_n$ be an optimal solution.
2. Pick r to be a uniformly distributed vector on the unit sphere S_n.
3. Let $S = \{v_i \mid a_i \cdot r \geq 0\}$ be the "true" variables.

Analysis

$$\max \sum_{0 \le i < j \le n} [a_{ij}(1 + v_i \cdot v_j) + b_{ij}(1 - v_i \cdot v_j)]$$

Term-by-term analysis.

First consider the second term.

Contribution to semidefinite program: $b_{ij}(1 - \cos\theta_{ij})$

Contribution to the solution: $2b_{ij} \cdot \mathbf{Pr}[v_i \text{ and } v_j \text{ are separated }] = 2b_{ij} \cdot \frac{\theta_{ij}}{\pi}$

Approximation Ratio: $\frac{2}{\pi} \cdot \frac{\theta_{ij}}{(1 - \cos\theta_{ij})} > 0.87856$

Analysis

$$\max \sum_{0 \le i < j \le n} [a_{ij}(1 + v_i \cdot v_j) + b_{ij}(1 - v_i \cdot v_j)]$$

Term-by-term analysis.

Consider the first term.

Contribution to semidefinite program: $a_{ij}(1 + \cos\theta_{ij})$

Contribution to the solution:

$$2a_{ij} \cdot \mathbf{Pr}[v_i \text{ and } v_j \text{ are NOT separated }] = 2a_{ij} \cdot (1 - \frac{\theta_{ij}}{\pi})$$

Approximation Ratio:

$$\frac{2}{\pi} \cdot \frac{\pi - \theta_{ij}}{(1 + \cos\theta_{ij})} = \frac{2}{\pi} \cdot \frac{\pi - \theta_{ij}}{(1 - \cos(\pi - \theta_{ij}))} > 0.87856$$

Algorithm

(MAX-2-SAT Algorithm)

1. Solve the vector program. Let $a_1, \ldots, a_n$ be an optimal solution.
2. Pick r to be a uniformly distributed vector on the unit sphere S_n.
3. Let $S = \{v_i \mid a_i \cdot r \geq 0\}$ be the "true" variables.

This is a 0.87856-approximation algorithm for MAX-2-SAT.

Can be improved to 0.931!

More SDP in approximation algorithms

- Sparsest cut: $O(\sqrt{\log n})$
- Graph colouring
- Constraint satisfaction problems
- Correlation clustering

Perhaps the most powerful weapon in the design of approximation algorithms.

Useful to know geometry and algebra.