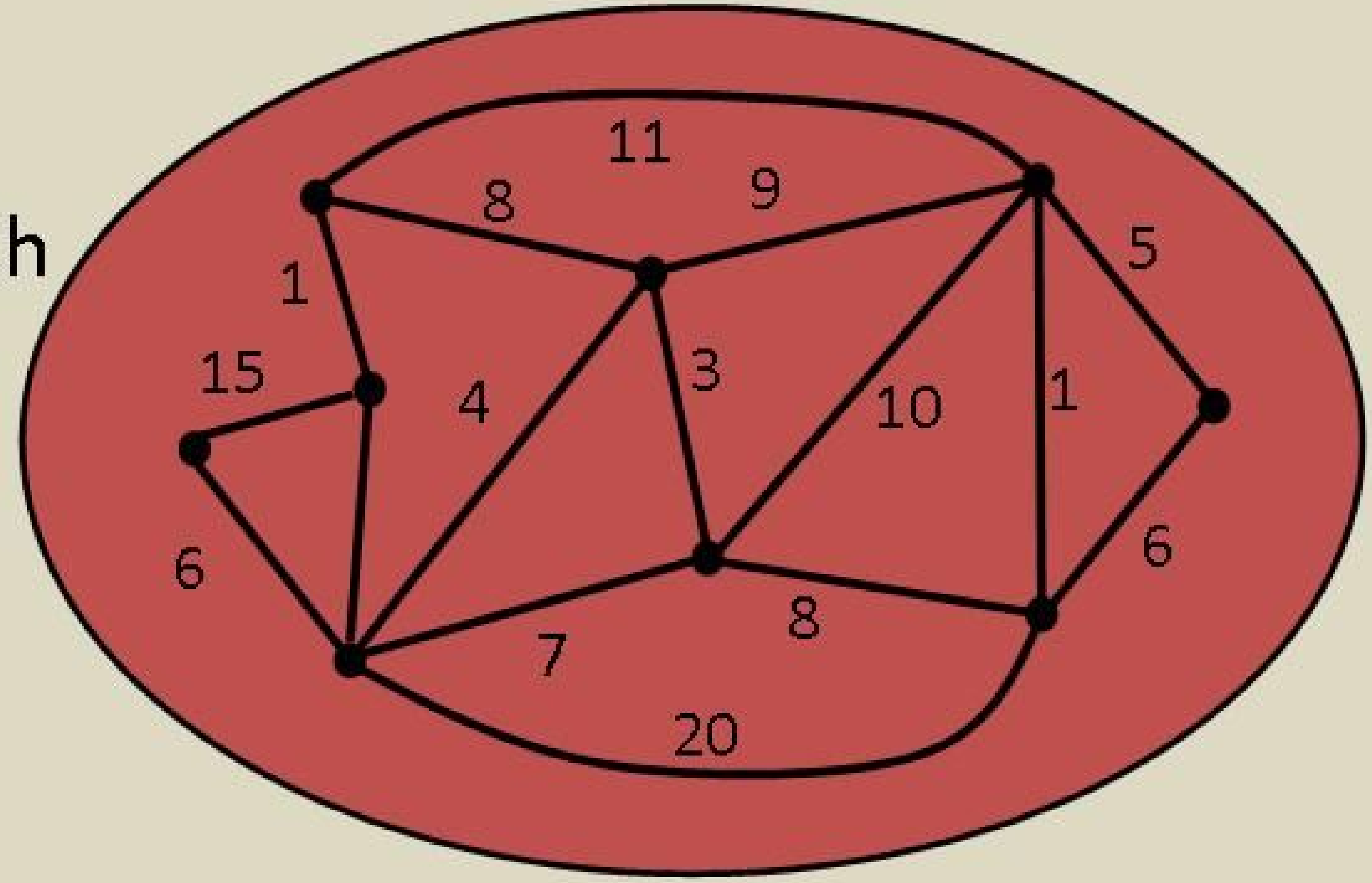
Fast approximation algorithms for cut-based problems in undirected graphs

Aleksander Mądry

MIT

The setup

Undirected graph **G** with integer capacities **u(·)**



What problems one might want to solve on **G**?

Popular choice:

Cut-based minimization problems

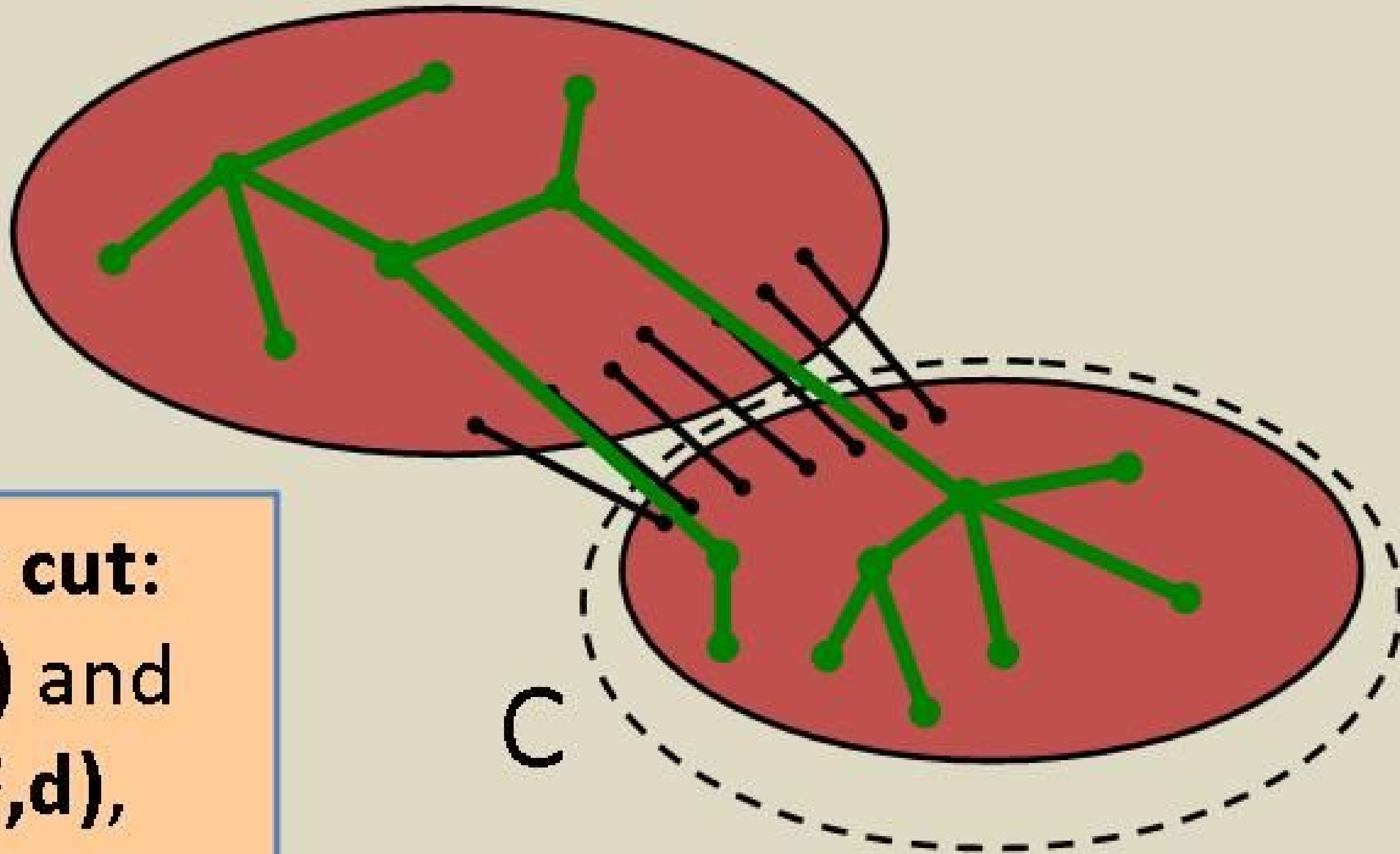
Examples of such problems

- **minimum cut problem**
- **minimum s-t cut problem**
- **(generalized) sparsest cut problem**
- **minimum conductance cut problem**
- **balanced separator problem**
- **minimum bisection problem**
- **…**

Motivation?

These problems are everywhere!

Our example



Generalized sparsest cut:
Given a graph **G=(V,E,u)** and a demand graph **D=(V,F,d)**, find a cut $\mathbf{C}^*$ that minimizes:

$$\frac{\mathbf{u(C)}}{\mathbf{d(C)}}$$

Applications:
Graph partitioning, bounding max concurrent flow ratio

Important special case:
If **D** is a complete graph → **uniform sparsest cut**

What questions are usually asked about such problems?

Can we solve them **efficiently**?

Is it in **P**?

No, they both are **NP**-hard

How well can we **approximate** them in poly-time?

Uniform sparsest cut : $O(\sqrt{\log n})$ **[Arora Rao Vazirani '04]**

Generalized sparsest cut : $O(\sqrt{\log n \log\log n})$ **[Arora Lee Naor '05]**

But there is **one more** question we should ask as well…

How well can we **approximate** them when we want to be **really efficient**?

Nearly-linear time

Rationale: This would be the **first** question asked when one wants to solve these problems **in practice**!

Note: we care a lot about real efficiency for problems in **P**, but not so much for the ones that are **NP-hard**

What is known in this context?

Uniform sparsest cut:

Spectral partitioning :

$O(\sqrt{n})$	$\tilde{O}(m)$	[Alon Milman '85][Andersen Peres '09]

Flow - based algorithms :

$O(\log n)$	$\tilde{O}(n^2)$	[Leighton Rao '99]
$O(\sqrt{\log n})$	$\tilde{O}(n^2)$	[Arora Hazan Kale '04]
$O(\log^2 n)$	$\tilde{O}(m+n^{3/2})$	[Khandekar Rao Vazirani '06]
$O(\log n)$	$\tilde{O}(m+n^{3/2})$	[Arora Kale '07]
$O(\log n)$	$\tilde{O}(m+n^{3/2})$	[Orecchia Schulman Vazirani Vishnoi '08]
$O(\sqrt{\log n/\varepsilon})$	$\tilde{O}(m+n^{3/2+\varepsilon})$	[Sherman '09]

Generalized sparsest cut:

$O(\log n)$	$\tilde{O}(n^2 \log U)$	[Leighton Rao '99]

Our result

Generalized sparsest cut:

For any integral **ε>0**, we can **α(ε)-approximate** the problem in $\tilde{O}(m+n^{1+\varepsilon})$ time with $\alpha(\varepsilon) \approx \log^{(\log 1/\varepsilon)} n$

$\tilde{O}(m+n^{(1+\varepsilon)}\varepsilon^{-1} \log U)$

$\log^{(1+o(1))\lceil 1+\log 1/\varepsilon \rceil} n$

k=1 → ($\log^{(1+o(1))} n$)-approx in $\tilde{O}(n^2 \log U)$ time

k=2 → ($\log^{(2+o(1))} n$)-approx in $\tilde{O}(m+n^{4/3} \log U)$ time

k=3 → ($\log^{(3+o(1))} n$)-approx in $\tilde{O}(m+n^{8/7} \log U)$ time

We get time **arbitrarily close to nearly-linear**, but **pay** accordingly in **approximation** guarantee

(even better trade-off for **uniform** version)

We can do even more!

Our result (cont.)

Let us call a **minimization** problem $\mathcal{P}$ **cut-based** if we can cast it as:
Given an instance **P** of $\mathcal{P}$ and **G=(V,E,u)**,
find a cut **C*** being $\mathbf{argmin_C\ u(C)f_P(C)}$,
where $\mathbf{f_P(C)}$ depends only on **P**

Examples:

Minimum s-t cut problem:

$\mathbf{f_P(C)= 1}$ if **C** separates **s** and **t**;
$\mathbf{f_P(C)= +\infty}$ otherwise.

Generalized sparset cut problem:

$\mathbf{f_P(C)= 1/d(C)}$

Our result (simplified)

For **any cut-based minimization** problem $\mathcal{P}$, given an algorithm **A** that **β-approximates** $\mathcal{P}$ only on **tree** instances, for any **ε>0**, we get an **(α(ε)·β)-approximation** for $\mathcal{P}$ on **general graphs** in time $\mathbf{\tilde{O}(m+n^{(1+\varepsilon)})+}$ time needed to run **A** on **≈1/ε** tree instances

Moral: When aiming at **(fast) poly-log approximation** of a **minimization cut-based problem:** just focus on **tree instances**

Example: On trees we can solve **generalized sparsest cut** optimally (in $\mathbf{\tilde{O}(m)}$ time) → our result follows

How to go about proving such theorem?

[Räcke '08] **(simplified):** For any graph **$G=(V,E,u)$**, we can find in poly-time a convex combination **$\{(\lambda_i,T_i)\}_i$** of **trees*** s.t. for any cut **C**:

(cut lower-bounding) $\mathbf{u_i(C) \geq u(C)}$ for all **i**

(cut upper-bounding) $\mathbf{E_\lambda[u(C)] := \sum_i \lambda_i u_i(C) \leq O(\log n)\, u(C)}$

Idea for lifting: Find **$\{(\lambda_i,T_i)\}_i$** as above and sample a tree **T** equal to **T_i** with probability **λ_i**
output an **α-optimal** solution **C** for instance **P** on tree **T**

Why should it work? Let **C*** be the optimal solution

with prob. **≥ 1/2:** $\mathbf{u_T(C^*) \leq O(\log n)\, u(C^*)}$

But $\mathbf{u(C)\, f_P(C) \leq u_T(C)\, f_P(C) \leq \alpha\, u_T(C^*) f_P(C^*) \leq}$

$\mathbf{O(\alpha \log n)\, u(C^*) f_P(C^*) = O(\alpha \log n)\, OPT}$

Note: Choice of **T** is **oblivious** to the problem we want to solve!

How to go about proving such theorem?

[Räcke '08] **(simplified):** For any graph **$G=(V,E,u)$**, we can find in poly-time a convex combination **$\{(\lambda_i,T_i)\}_i$** of **trees*** s.t. for any cut **C**:

(cut lower-bounding) $u_i(C) \geq u(C)$ for all i

(cut upper-bounding) $E_\lambda[u(C)] := \sum_i \lambda_i u_i(C) \leq O(\log n)\, u(C)$

Idea for lifting: Find $\{(\lambda_i,T_i)\}_i$ as above and sample a tree **T** equal to **T_i** with probability **λ_i**

output an **α-optimal** solution **C** for instance **P** on tree **T**

Lifting works great! How about running time?

Räcke's algorithm runs in **$\tilde{O}(m \min(mn, n^\omega))$** time

Prohibitive from our point of view!

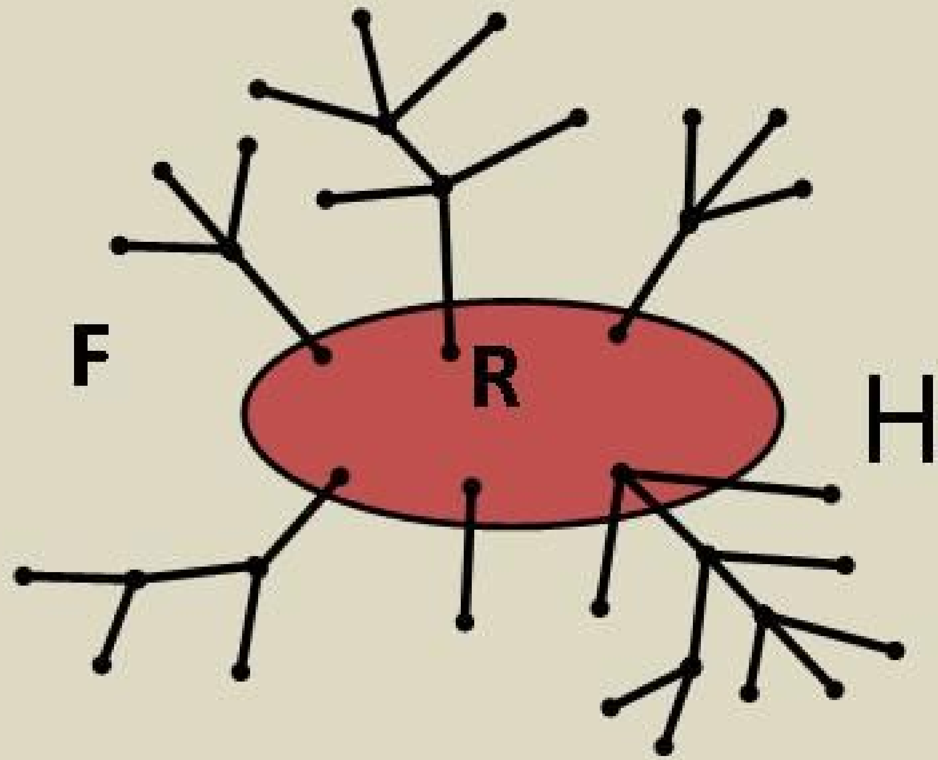
Can speed it up to **$\tilde{O}(m^2)$** time while losing a bit in quality

But this is still not enough!

What to do now?
The approach looked very promising but got stuck…
Maybe we are asking for too much?

Idea: Decompose **G** into objects that are **more complicated than trees**, **but** still **simpler than general graphs**

H is a **j-tree** if it is a union of:
→ forest **F** (**envelope**)
→ arbitrary graph **R** (**core**)
and:
1) **$|V(R)| \leq j$**
2) for each connected component **F'** of **F**, **$|V(F') \cap V(R)| = 1$**



Note: 1-tree is just a **tree**

Decomposing graphs into **j-trees**

Theorem (simplified): For any graph **$G=(V,E,u)$** and **$j\geq 1$**, we can find in $\tilde{O}(m^2/j)$ a convex combination $\{(\lambda_i,T_i)\}_i$ of **j-trees** s.t. for any cut **C**:

(cut lower-bounding) $u_i(C) \geq u(C)$ for all **i**

(cut upper-bounding) $E_\lambda[u(C)] := \sum_i \lambda_i u_i(C) \leq \tilde{O}(\log n)\, u(C)$

If we tak **j=1** then we recover Räcke's result with **faster** running time, but slightly **worse** quality

But the ability to vary **j** gives a lot of **flexibility**!

Rough intuition: The "real" complexity of a **cut-based** problem on a **j-tree** with **n** vertices is **j** not **n**

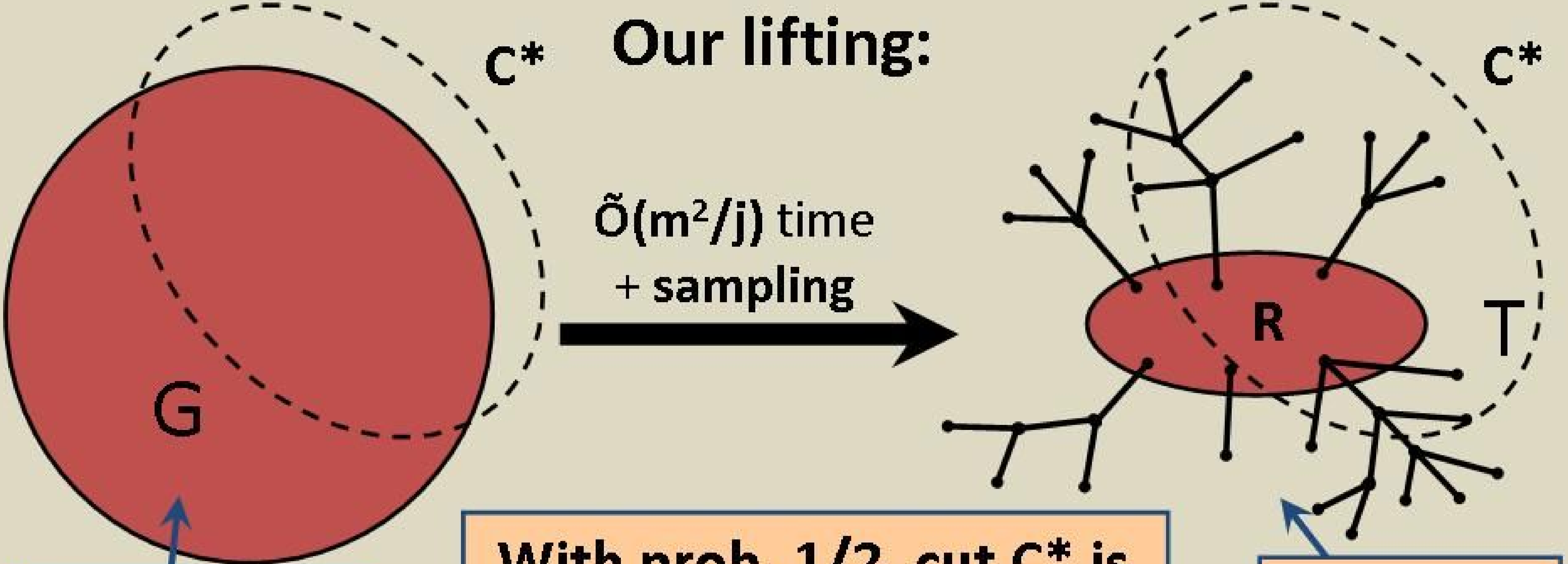
If **A** works in $\tilde{O}(m+n^{(1+c)})$ time on general graphs

↓**(heuristically)**↓

It can be made to work in $\tilde{O}(m+j^{(1+c)})$ time on **j**-trees

This allows **speeding up** such algorithms!

Our lifting:



With prob. 1/2, cut C* is Õ(log n)-preserved

We now run our algorithm on **T** instead of **G** to get a speed up!

But there is even a better way of leveraging this flexibility!

Instead of reducing **G** to **T** in one **big step**…
…we do it in a series of **small recursive steps**

We get a running time **arbitrarily** close to **nearly-linear**

…but at a **price** of **approximation ratio growing** accordingly

Conclusions and open problems

We presented a general method of obtaining **fast poly-log approximation** algorithms for **minimization cut-based** problems

(Our method is oblivious to actual problem we want to solve)

Can one get a **better trade-off**?

Maybe some **fixed** poly-log approximation in **nearly-linear time**?

...at least for some specific problem (e.g. sparsest cut)?

Can one extend this method to **flow problems**?

Key take-away question: How well can we **approximate fundamental problems** while being **really efficient**?

Thank you!

Questions?