Approximating CSPs in more than polynomial time

Michael Lampis
Université Paris Dauphine

May 31, 2018

ESIGMA Kick-off Meeting

Overview

Things you will hear in this talk:

- Max-CSPs such as Max-SAT, Max-Cut, etc. are Hard!
- . . . even to approximate!
- To solve them we need to:
  - Take into account the input stucture (how?)
  - Invest a little more than polynomial time (how much?)
  - Allow some sub-optimal solutions (but almost optimal!)

Overview

Things you will hear in this talk:

- Max-CSPs such as Max-SAT, Max-Cut, etc. are Hard!
- . . . even to approximate!
- To solve them we need to:
  - Take into account the input stucture (how?)
  - Invest a little more than polynomial time (how much?)
  - Allow some sub-optimal solutions (but almost optimal!)

Results based on two papers:

- ”Sub-Exponential Approximation Schemes for CSPs: From Dense to Almost-Sparse.”, Dimitris Fotakis, Michael Lampis, and Vangelis Th. Paschos, STACS ’16.
- ”Complexity and Approximability for Parameterized CSPs.”, Holger Dell, Eunjung Kim, Michael Lampis, Valia Mitsou, Tobias Moemke, IPEC’15 (Algorithmica ’17).

The Big Picture

The Big Picture

- We want to solve NP-hard optimization problems
  - . . . in this talk: Max-SAT, Max-Cut, Max-CSP in general

The Big Picture

- We want to solve NP-hard optimization problems
  - …in this talk: Max-SAT, Max-Cut, Max-CSP in general

The Big Picture

- We want to solve NP-hard optimization problems
  - ...in this talk: Max-SAT, Max-Cut, Max-CSP in general

Definition of “**Solve**”:

- Time-efficiency (polynomial time)
- Optimality
- Generality (handles all instances)

The Big Picture

- We want to solve NP-hard optimization problems
  - ... in this talk: Max-SAT, Max-Cut, Max-CSP in general

Definition of "**Solve**":

- Time-efficiency (polynomial time)
- Optimality
- Generality (handles all instances)

Main Challenge: Under standard complexity assumptions (P$\neq$NP, ETH), no algorithm achieves all three!

- In fact, tight hardness known for many problems:
- Max-3-SAT cannot be $(7/8-\epsilon)$-approximated, cannot be solved in $2^{o(n)}$.

Research Direction:

- Trade **Time** for **Generality** and/or **Optimality**

Dead on Arrival?

- Better than $3/2$ for TSP, $4/3$ for Max-3-DM, $7/8$ for Max-3-SAT,..., in **sub-exponential time**?

Dead on Arrival?

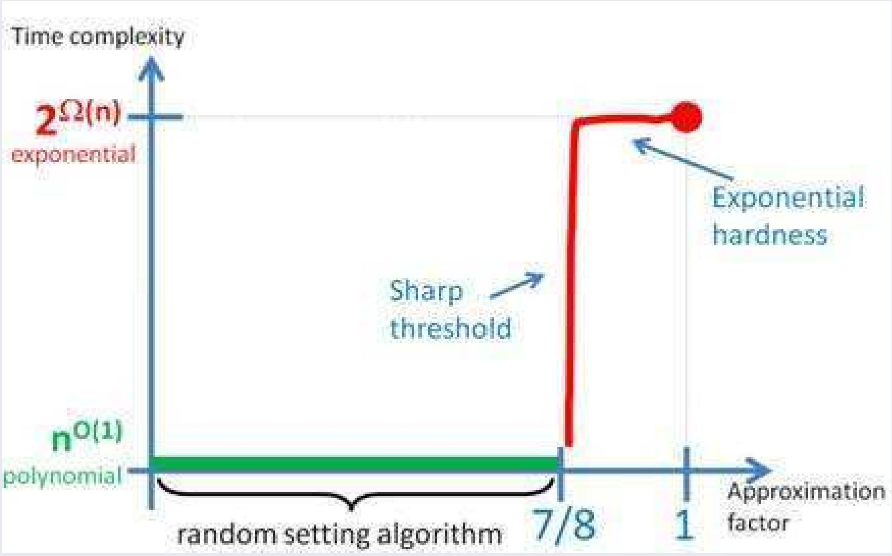
- Better than $3/2$ for TSP, $4/3$ for Max-3-DM, $7/8$ for Max-3-SAT,. . . , in **sub-exponential time**?

Probably won't work

(at least for Max-3-SAT)

Dead on Arrival?

- Better than $3/2$ for TSP, $4/3$ for Max-3-DM, $7/8$ for Max-3-SAT,..., in **sub-exponential time**?



Almost-linear PCPs (Moshkovitz& Raz) and P-time hardness (Håstad) give tight inapproximability for Max-3-SAT even for $2^{n^{1-\epsilon}}$ time.

(Credit: Dana Moshkovitz)

Dead on Arrival?

- Better than $3/2$ for TSP, $4/3$ for Max-3-DM, $7/8$ for Max-3-SAT,. . . , in **sub-exponential time**?

  If this is the “normal” behavior of APX problems, what’s the point of sub-exponential approximation?

  - Is this the “normal” behavior?
  - **What else can we do?**

Strategy

- We **cannot** get better than 7/8 for Max-3-SAT in sub-exp time (under ETH).
- We will therefore try to get something else:

Strategy

- We **cannot** get better than 7/8 for Max-3-SAT in sub-exp time (under ETH).
- We will therefore try to get something else:

An island of tractability:

- Max-k-CSP admits a PTAS (a $(1-\epsilon)$-approximation for all $\epsilon > 0$) for **dense** instances
- (Arora, Karger, Karpinski '99), (Fernandez de la Vega '96)

Strategy

- We **cannot** get better than 7/8 for Max-3-SAT in sub-exp time (under ETH).
- We will therefore try to get something else:

An island of tractability:

- Max-k-CSP admits a PTAS (a $(1-\epsilon)$-approximation for all $\epsilon > 0$) for **dense** instances
- (Arora, Karger, Karpinski '99), (Fernandez de la Vega '96)

Extending the island:

- We give a version of the AKK scheme which **can handle sparser instances**, at the expense of needing **sub-exponential time**.
- Our scheme provides a smooth trade-off
  - For dense instances we get a PTAS
  - As instances gradually get more sparse, we need more time…
  - …until our scheme does not work any more

Summary of results

For any $\epsilon > 0$, $\delta \in [0, 1]$ and fixed $k \geq 2$ we have the following:

- Given a Max-k-CSP instance with $n^{k-1+\delta}$ constraints
- We can produce a $(1 - \epsilon)$-approximate solution
- In time $2^{O(n^{1-\delta} \ln n / \epsilon^3)}$

Summary of results

For any $\epsilon > 0$, $\delta \in [0, 1]$ and fixed $k \geq 2$ we have the following:

- Given a Max-k-CSP instance with $n^{k-1+\delta}$ constraints
- We can produce a $(1 - \epsilon)$-approximate solution
- In time $2^{O(n^{1-\delta} \ln n/\epsilon^3)}$

- Note: This includes the AKK PTAS as a special case ($\delta = 1$)
- Advantage: we provide a smooth trade-off from the "easy case" (dense instances) to more general cases

Summary of results

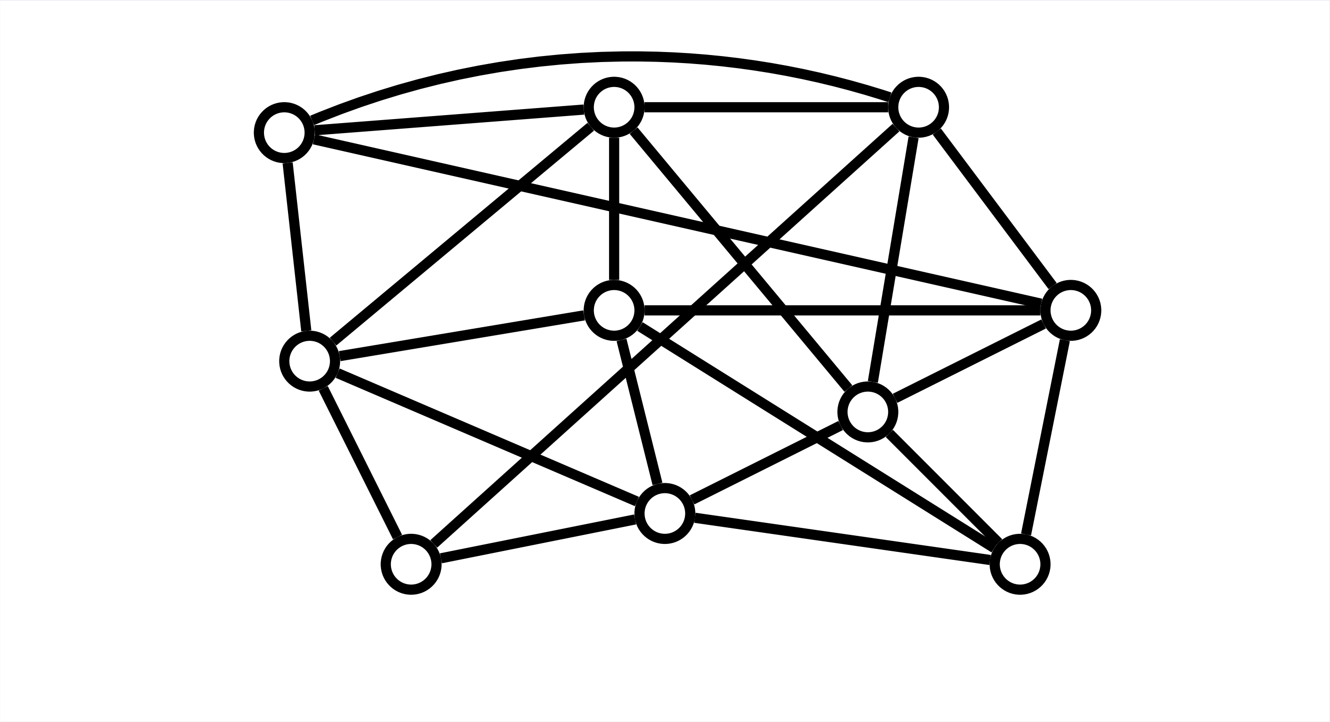
For any $\epsilon > 0$, $\delta \in [0, 1]$ and fixed $k \geq 2$ we have the following:

- Given a Max-k-CSP instance with $n^{k-1+\delta}$ constraints
- We can produce a $(1-\epsilon)$-approximate solution
- In time $2^{O(n^{1-\delta} \ln n / \epsilon^3)}$

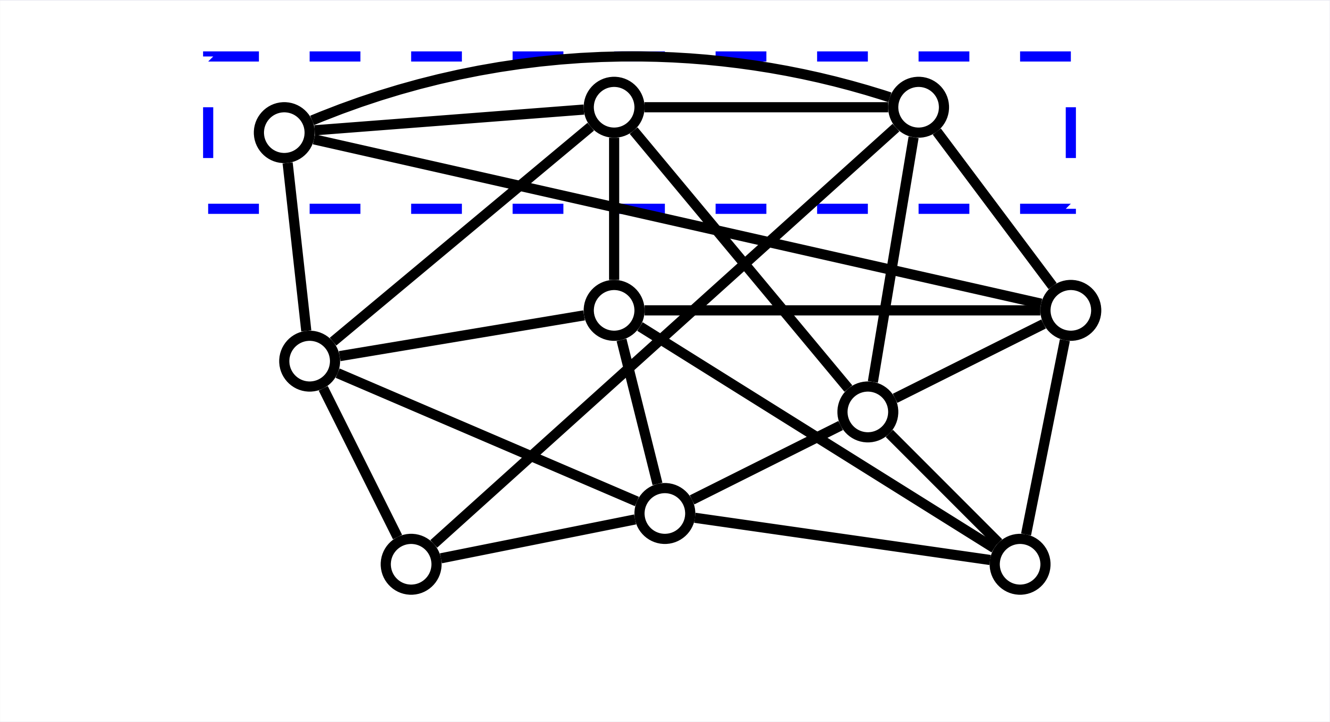
- Note: This includes the AKK PTAS as a special case ($\delta = 1$)
- Advantage: we provide a smooth trade-off from the "easy case" (dense instances) to more general cases

- We will also give some "tight" bounds, ruling out natural possible improvements.

Basic scheme (Max Cut)

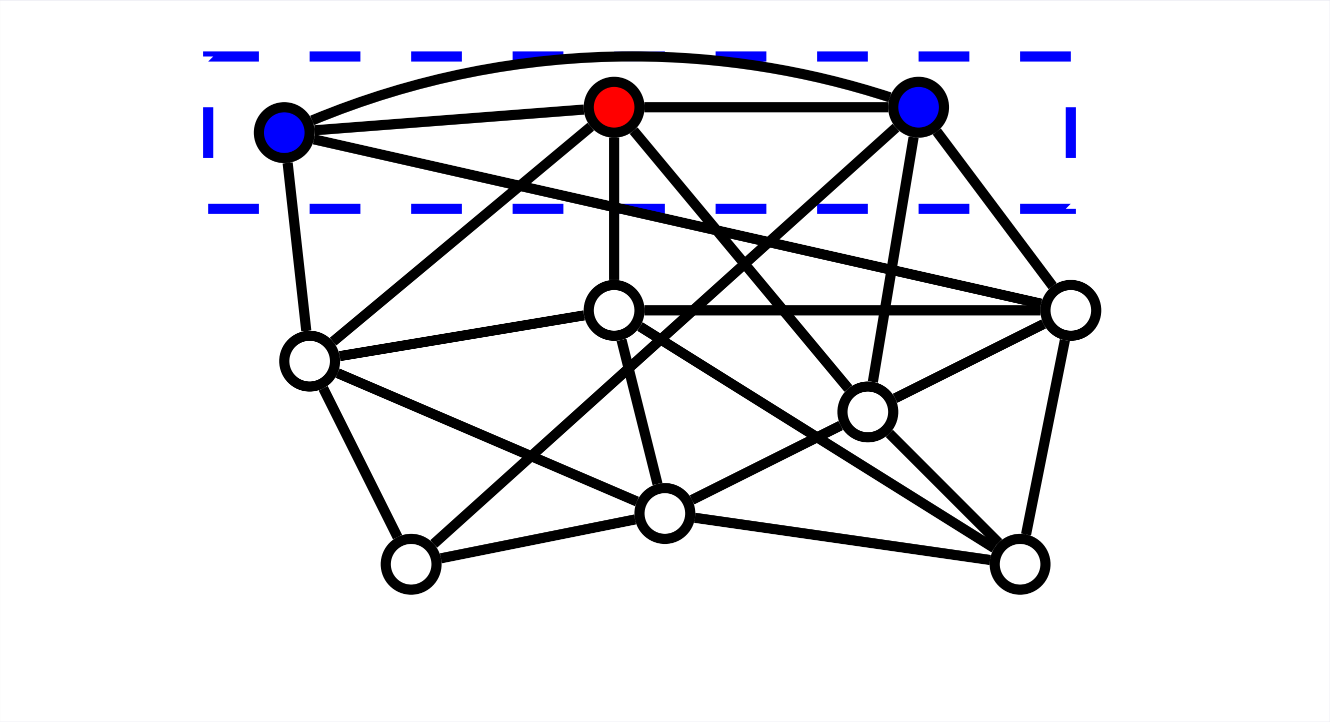


We are given a dense graph for which we want to find a large cut



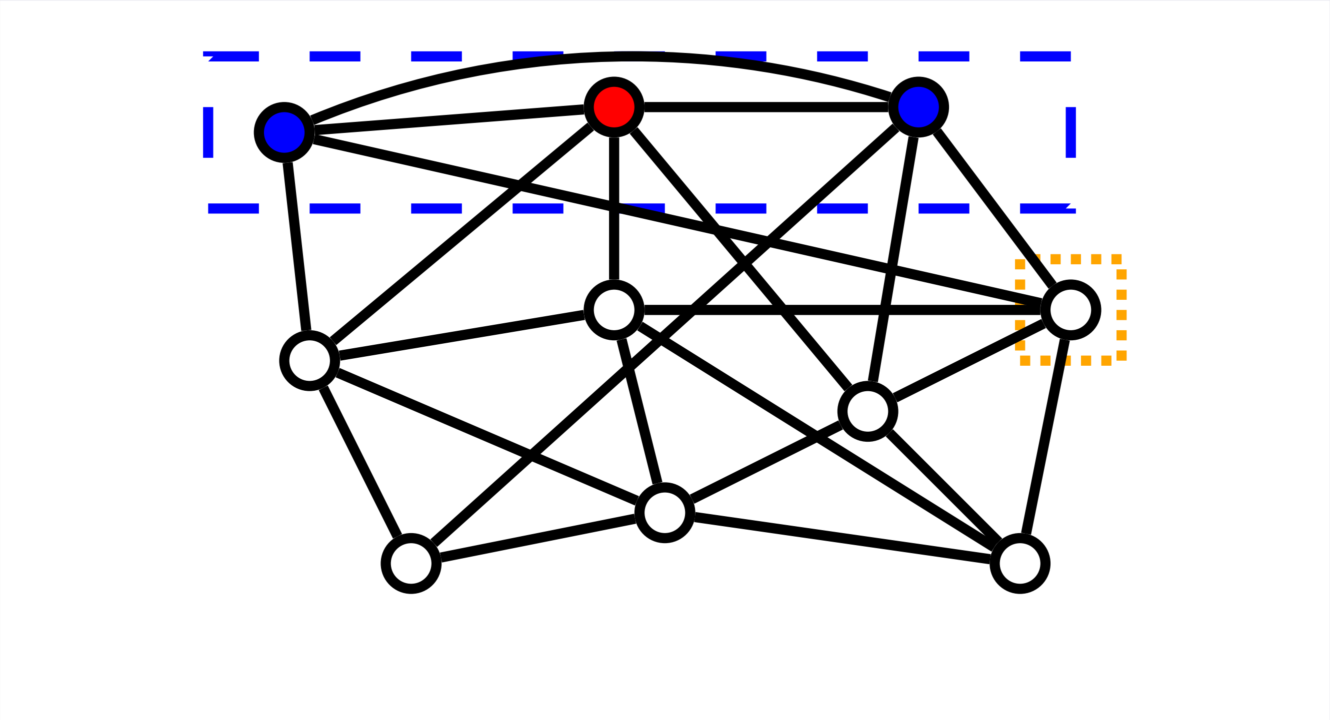
Randomly select a “sample” of its vertices

Basic scheme (Max Cut)



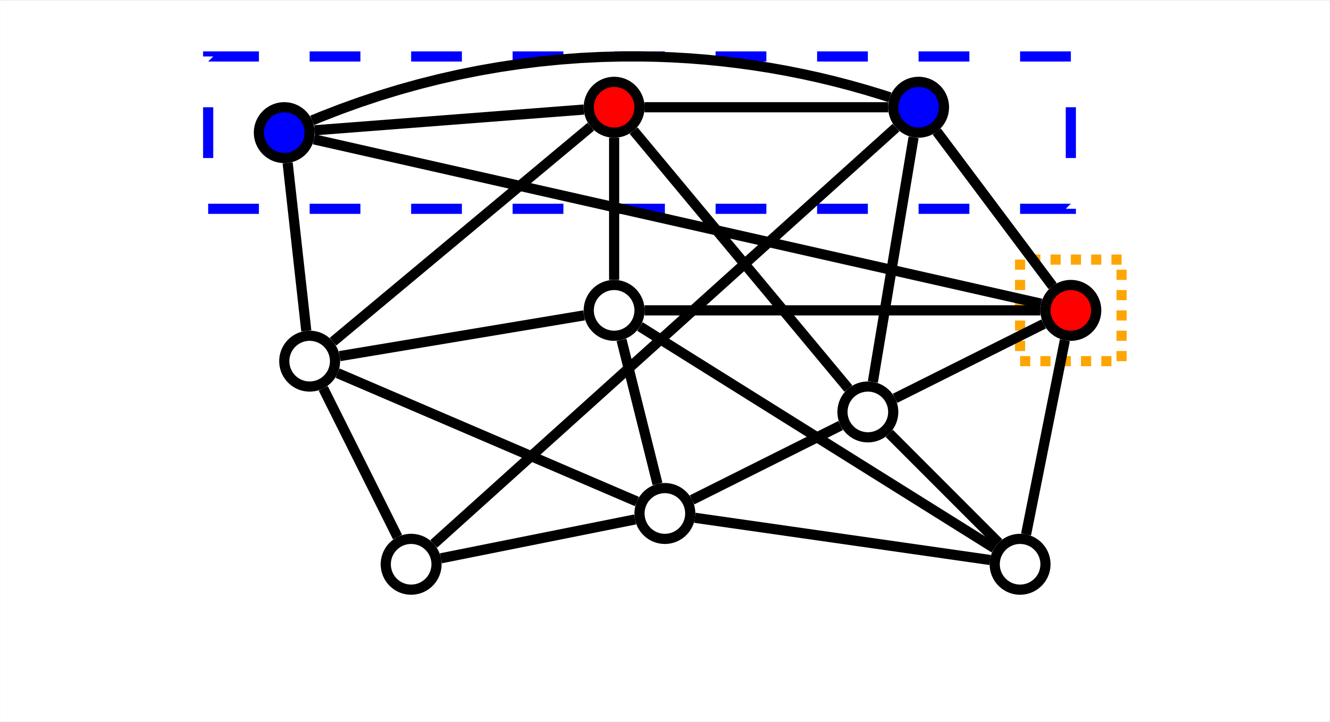
Guess their correct partition

Basic scheme (Max Cut)



For every vertex outside the sample, examine its neighbors in the sample

Basic scheme (Max Cut)



Greedily set its value depending on this neighborhood

Basic scheme (Max Cut)

- The sample we select has size $O(\log n)$ (hidden constants depend on degree and ϵ)
- $\rightarrow$ running time $n^{O(1)}$ (will try all partitions of sample)

Basic scheme (Max Cut)

- The sample we select has size $O(\log n)$ (hidden constants depend on degree and ϵ)
- $\rightarrow$ running time $n^{O(1)}$ (will try all partitions of sample)

Why this works (intuitively):

- Because graph is dense $\rightarrow$ every vertex outside sample S has many neighbors in S
- $\rightarrow$ examining $N(u) \cap S$ is (whp) a good representation of $N(u)$ in the optimal solution
- If a vertex in $V \setminus S$ has $>> 50\%$ of its neighbors on one side in the optimal solution, it will (whp) have $>> 50\%$ of its neighbors on that side in S

(Fernandez de la Vega '96)

General scheme (Max-k-CSP)

Max Cut:

$$\max \sum_{(i,j)\in E} x_i(1-x_j) + x_j(1-x_i)$$

General scheme (Max-k-CSP)

Max-2-SAT:

$$\max \sum_{(i,j)\in C} x_i(1-x_j) + x_j(1-x_i) + x_i x_j$$

General scheme (Max-k-CSP)

Max-3-SAT:

$$\max \sum_{(i,j,k)\in C} x_i(1-x_j)(1-x_k) + (1-x_i)x_j(1-x_k) + \ldots + x_i x_j x_k$$

General scheme (Max-k-CSP)

Max-k-CSP:

$$\max p(\vec{x})$$

where $p()$ is a degree k polynomial.

The AKK scheme offers a PTAS that finds an assignment almost maximizing p when the polynomial has at least $\Omega(n^k)$ terms.

General scheme (Max-k-CSP)

Max Cut:

$$\max \sum_{(i,j)\in E} x_i(1-x_j) + x_j(1-x_i)$$

General scheme (Max-k-CSP)

Max Cut:

$$\max \sum_{(i,j)} c_{ij} x_i x_j + \sum_i c_i x_i + C$$

General scheme (Max-k-CSP)

Max Cut:

$$\max \sum_i x_i r_i$$

where $r_i(\vec{x} - x_i)$ is the (linear) polynomial of the remaining variables I obtain if I factor out x_i.

General scheme (Max-k-CSP)

Max Cut:

$$\max \sum_i x_i r_i$$

where $r_i(\vec{x} - x_i)$ is the (linear) polynomial of the remaining variables I obtain if I factor out x_i.

Main idea: Estimate the values of the r_i's using brute force on a small sample.

General scheme (Max-k-CSP)

Max Cut:

$$\max \sum_i x_i r_i$$

s.t.

$$\hat{r}_i - \epsilon n \leq \quad \textstyle\sum_{j \in N(i)} c_{ij} x_j \quad \leq \hat{r}_i + \epsilon n$$

where $\hat{r}_i$ is the estimate I have for r_i.

This is now a **linear** program.

General scheme (Max-k-CSP)

Max Cut:

$$\max \sum_i x_i r_i$$

Summary of algorithm:

- Estimate the r_i values using a sample
  - Need large enough sample to guarantee $\hat{r}_i \approx r_i$
  - This turns QIP $\rightarrow$ ILP
- Solve fractional relaxation of ILP
- Round solution

Sub-exponential Extension (Max Cut)

Summary of algorithm:

- Estimate the r_i values using a sample
  - Need large enough sample to guarantee $\hat{r}_i \approx r_i$
  - This turns QIP $\rightarrow$ ILP
- Solve fractional relaxation of ILP
- Round solution

Sub-exponential Extension (Max Cut)

Summary of algorithm:

- Estimate the r_i values using a sample
  - Need large enough sample to guarantee $\hat{r}_i \approx r_i$
  - This turns QIP $\rightarrow$ ILP
- Solve fractional relaxation of ILP
- Round solution

Main idea: **Use larger sample**

Sub-exponential Extension (Max Cut)

Summary of algorithm:

- Estimate the r_i values using a sample
  - Need large enough sample to guarantee $\hat{r}_i \approx r_i$
  - This turns QIP $\rightarrow$ ILP
- Solve fractional relaxation of ILP
- Round solution

Main idea: **Use larger sample**

- Suppose graph has average degree $\Delta = n^\delta$
- We sample $\frac{n \log n}{\Delta} = n^{1-\delta} \log n$ vertices
- $\rightarrow$ whp $\hat{r}_i \approx r_i$.

Sub-exponential Extension (Max Cut)

Summary of algorithm:

- Estimate the r_i values using a sample
  - Need large enough sample to guarantee $\hat{r}_i \approx r_i$
  - This turns QIP $\rightarrow$ ILP
- Solve fractional relaxation of ILP
- Round solution

Main idea: **Use larger sample**
We are almost done!

- Must prove sample size enough for $\hat{r}_i$
  - Pitfall: Additive error ϵn no longer negligible!
- Must prove rounding step still works

Don't worry, it all works!

General scheme $k \geq 3$

Summary so far $k = 2$:

- AKK: Average degree $\Omega(n)$, sample of $O(\log n)$ vertices
- Extension: Average degree n^δ, sample of $n^{1-\delta} \log n$
- $\rightarrow$ in time $2^{\sqrt{n}}$ can "solve" Max-Cut for $|E| \geq n^{1.5}$

General scheme $k \geq 3$

Summary so far $k = 2$:

- AKK: Average degree $\Omega(n)$, sample of $O(\log n)$ vertices
- Extension: Average degree n^{δ}, sample of $n^{1-\delta} \log n$
- $\rightarrow$ in time $2^{\sqrt{n}}$ can "solve" Max-Cut for $|E| \geq n^{1.5}$

How about Max-3-SAT?

- In poly time can solve instances with n^3 clauses
- In $2^{\sqrt{n}}$ time can solve instances with $\ldots$ clauses?

General scheme $k \geq 3$

Summary so far $k = 2$:

- AKK: Average degree $\Omega(n)$, sample of $O(\log n)$ vertices
- Extension: Average degree n^{δ}, sample of $n^{1-\delta} \log n$
- $\rightarrow$ in time $2^{\sqrt{n}}$ can "solve" Max-Cut for $|E| \geq n^{1.5}$

How about Max-3-SAT?

- In poly time can solve instances with n^3 clauses
- In $2^{\sqrt{n}}$ time can solve instances with $n^{2.5}$ clauses

General scheme $k \geq 3$

AKK scheme for $k \geq 3$

- Write $p(\vec{x})$ as $\sum_i x_i r_i$
- Each r_i has degree $k-1$
- Write $r_i = \sum_j x_j r_{ij}$
- Each r_{ij} has degree $k-2$
- $\ldots$
- Until we get to linear $\rightarrow$ write ILP

General scheme $k \geq 3$

AKK scheme for $k \geq 3$

- Write $p(\vec{x})$ as $\sum_i x_i r_i$
- Each r_i has degree $k-1$
- Write $r_i = \sum_j x_j r_{ij}$
- Each r_{ij} has degree $k-2$
- $\ldots$
- Until we get to linear $\rightarrow$ write ILP

Note: In order for this to work, all $r_{ij\ldots}$ polynomials must be **dense**

- This is true if original polynomial was dense.

General scheme $k \geq 3$

AKK scheme for $k \geq 3$

- Write $p(\vec{x})$ as $\sum_i x_i r_i$
- Each r_i has degree $k-1$
- Write $r_i = \sum_j x_j r_{ij}$
- Each r_{ij} has degree $k-2$
- ...
- Until we get to linear $\rightarrow$ write ILP

- In our scheme, if p has $n^{k-1+\delta}$ terms
- r_i has $n^{k-2+\delta}$ terms
- r_{ij} has $n^{k-3+\delta}$ terms
- ...

It seems that the “right” density to require is $n^{k-1+\delta}$?

General scheme – summary

- Input: Max-k-CSP instance with $n^{k-1+\delta}$ constraints
- Algorithm:
  - Sample $2^{n^{1-\delta}} \log n/\epsilon^3$ variables, guess their value
  - Write CSP as a polynomial optimization problem
  - Estimate non-linear coefficients using sample
  - Solve fractional LP
  - Round solution

General scheme – summary

- Input: Max-k-CSP instance with $n^{k-1+\delta}$ constraints
- Algorithm:
  - Sample $2^{n^{1-\delta}\log n/\epsilon^3}$ variables, guess their value
  - Write CSP as a polynomial optimization problem
  - Estimate non-linear coefficients using sample
  - Solve fractional LP
  - Round solution

- Works for any CSP (for fixed k)
- Covers "all instances" for $k = 2$

General scheme – summary

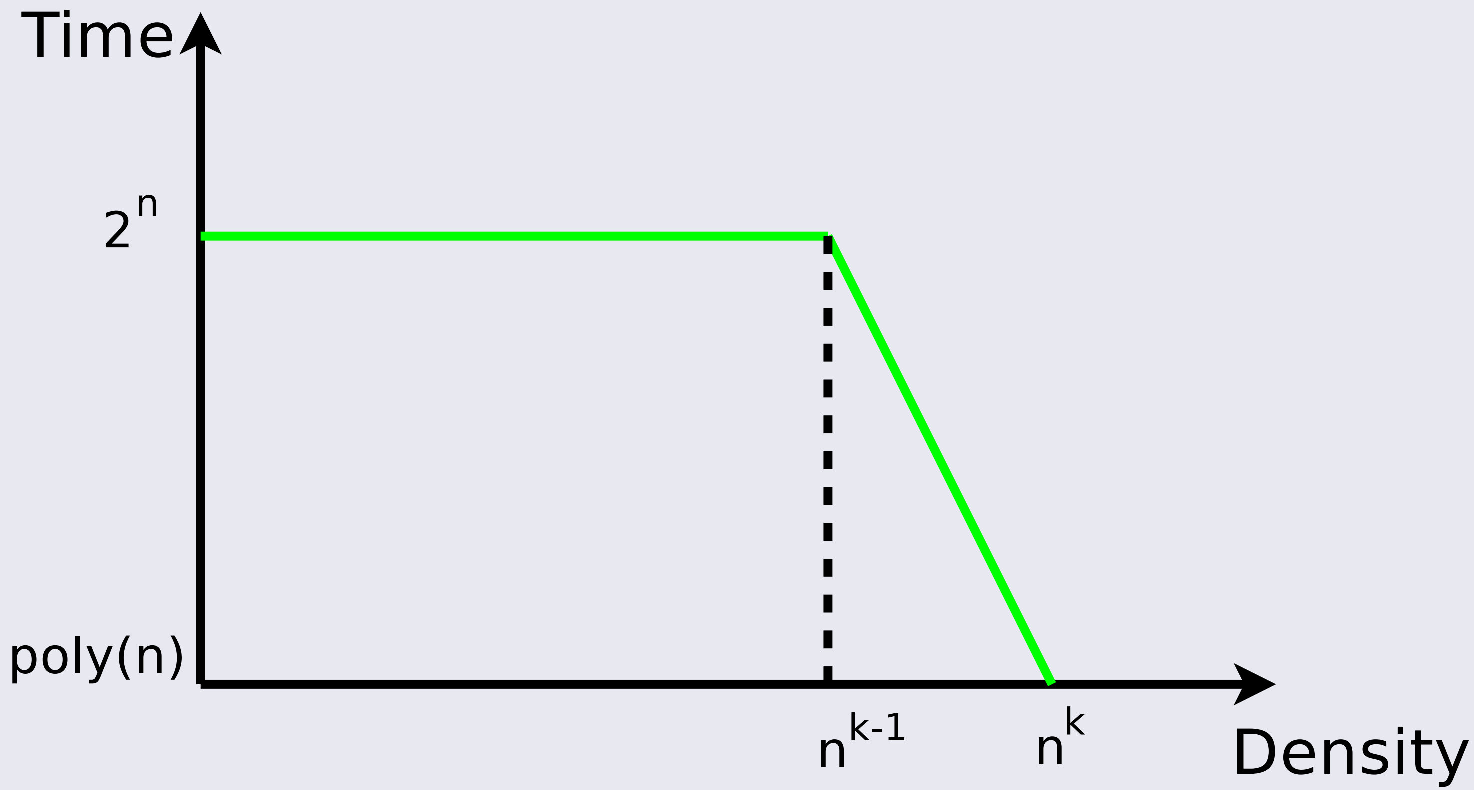
- Input: Max-k-CSP instance with $n^{k-1+\delta}$ constraints
- Algorithm:
  - Sample $2^{n^{1-\delta}\log n/\epsilon^3}$ variables, guess their value
  - Write CSP as a polynomial optimization problem
  - Estimate non-linear coefficients using sample
  - Solve fractional LP
  - Round solution

- Works for any CSP (for fixed k)
- Covers "all instances" for $k = 2$

Can we do better?

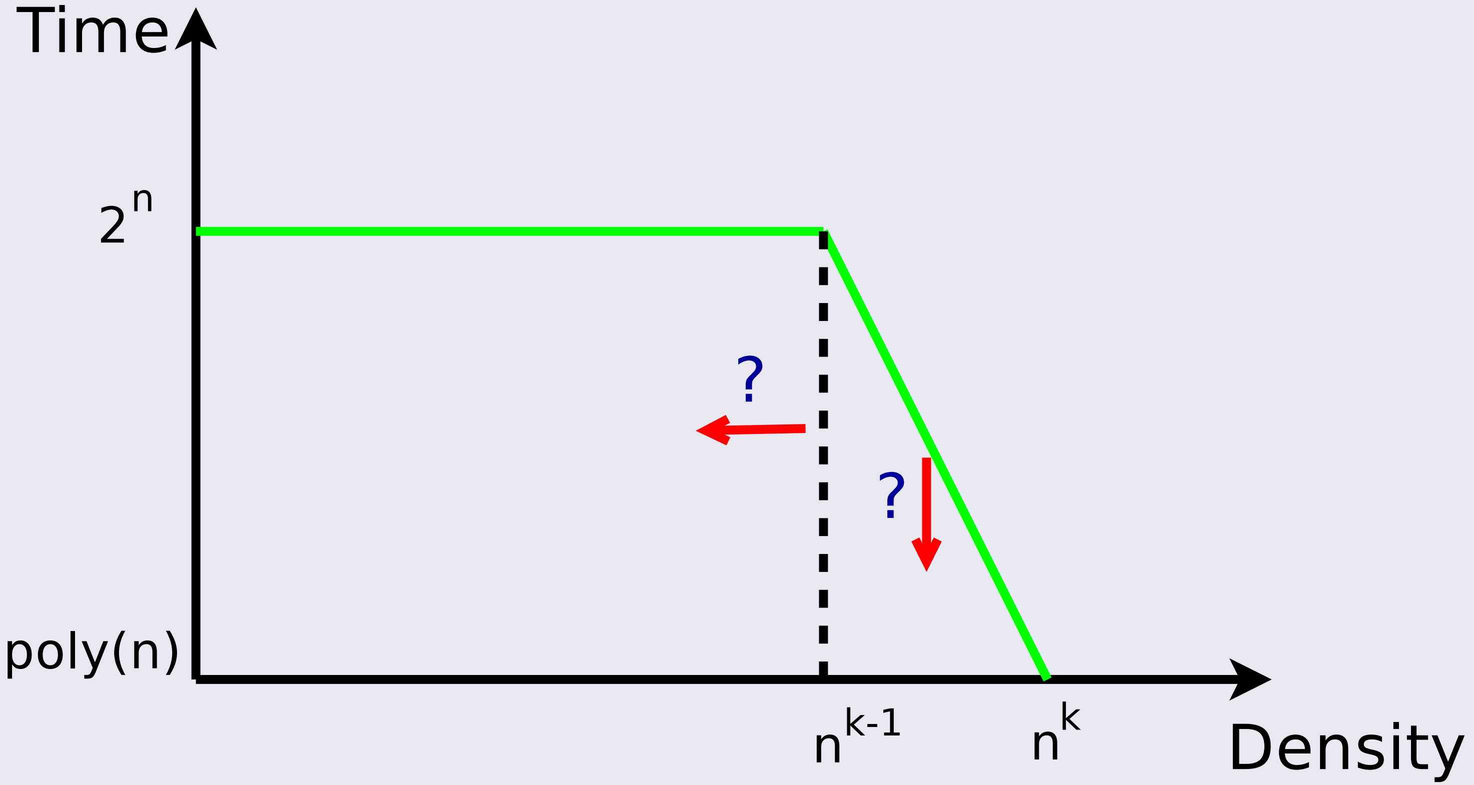
- Smaller sample/faster running time?
- Handle $k \geq 3$ better?

Summary – with a picture



Complexity/Density trade-off for Max-k-CSP

Summary – with a picture



Possible Improvements? Faster? More general?

Density lower bound

Theorem: Assuming ETH, for all $k \geq 3$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max-k-SAT with $m \leq n^{k-1}$ in time $2^{n^{1-\epsilon}}$

Density lower bound

Theorem: Assuming ETH, for all $k \geq 3$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max-k-SAT with $m \leq n^{k-1}$ in time $2^{n^{1-\epsilon}}$

In English: For density less than n^{k-1} we need exponential time to get $(1-\epsilon)$-approximation.

Density lower bound

Theorem: Assuming ETH, for all $k \geq 3$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max-k-SAT with $m \leq n^{k-1}$ in time $2^{n^{1-\epsilon}}$

Starting Point: Max-2-SAT is “APX-ETH”-hard on instances with $|V| = n$ and $m = O(|V|)$.

Density lower bound

Theorem: Assuming ETH, for all $k \geq 3$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max-k-SAT with $m \leq n^{k-1}$ in time $2^{n^{1-\epsilon}}$

Starting Point: Max-2-SAT is "APX-ETH"-hard on instances with $|V| = n$ and $m = O(|V|)$.
Proof: ($k = 3$)

- Add n new variables $y_1, \ldots, y_n$
- For each clause $(x_i \vee x_j)$, for each $k \in \{1, \ldots, n\}$ we construct the clauses $(x_i \vee x_j \vee y_k)$ and $(x_i \vee x_j \vee \neg y_k)$
- Gap remains!
- Number of clauses $\approx n^2$

Density lower bound

Theorem: Assuming ETH, for all $k \geq 3$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max-k-SAT with $m \leq n^{k-1}$ in time $2^{n^{1-\epsilon}}$

Starting Point: Max-2-SAT is "APX-ETH"-hard on instances with $|V| = n$ and $m = O(|V|)$.
Proof: ($k = 3$)

- Add n new variables $y_1, \ldots, y_n$
- For each clause $(x_i \vee x_j)$, for each $k \in \{1, \ldots, n\}$ we construct the clauses $(x_i \vee x_j \vee y_k)$ and $(x_i \vee x_j \vee \neg y_k)$
- Gap remains!
- Number of clauses $\approx n^2$

Reduction similar for $k > 3$

Running time lower bound

Theorem: Assuming ETH, for all $\delta \in (0,1)$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max Cut with $|E| = n^{1+\delta}$ in time $2^{n^{1-\delta-\epsilon}}$

Running time lower bound

Theorem: Assuming ETH, for all $\delta \in (0,1)$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max Cut with $|E| = n^{1+\delta}$ in time $2^{n^{1-\delta-\epsilon}}$

In English: Our sample size is optimal. For density n^{δ} we need time $2^{n^{1-\delta}}$.

Running time lower bound

Theorem: Assuming ETH, for all $\delta \in (0,1)$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max Cut with $|E| = n^{1+\delta}$ in time $2^{n^{1-\delta-\epsilon}}$

Starting Point: Max Cut is "APX-ETH"-hard on instances with $|V| = n$ and $|E| = O(|V|)$.

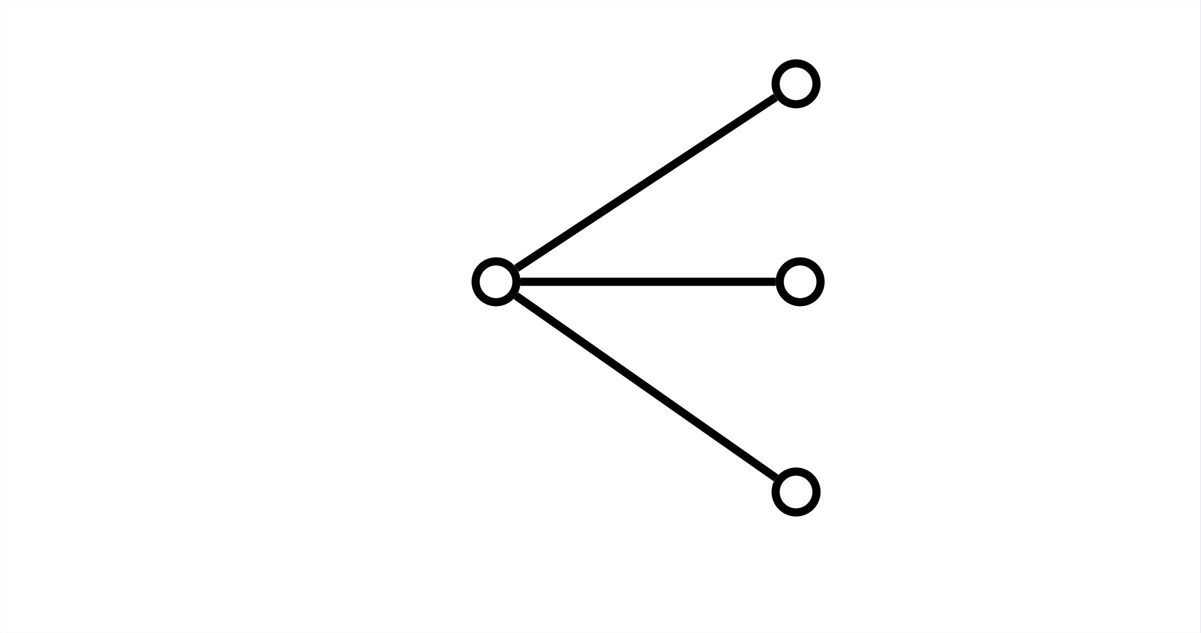
Running time lower bound

Theorem: Assuming ETH, for all $\delta \in (0,1)$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max Cut with $|E| = n^{1+\delta}$ in time $2^{n^{1-\delta-\epsilon}}$

Starting Point: Max Cut is "APX-ETH"-hard on 5-regular instances with $|V| = n$.

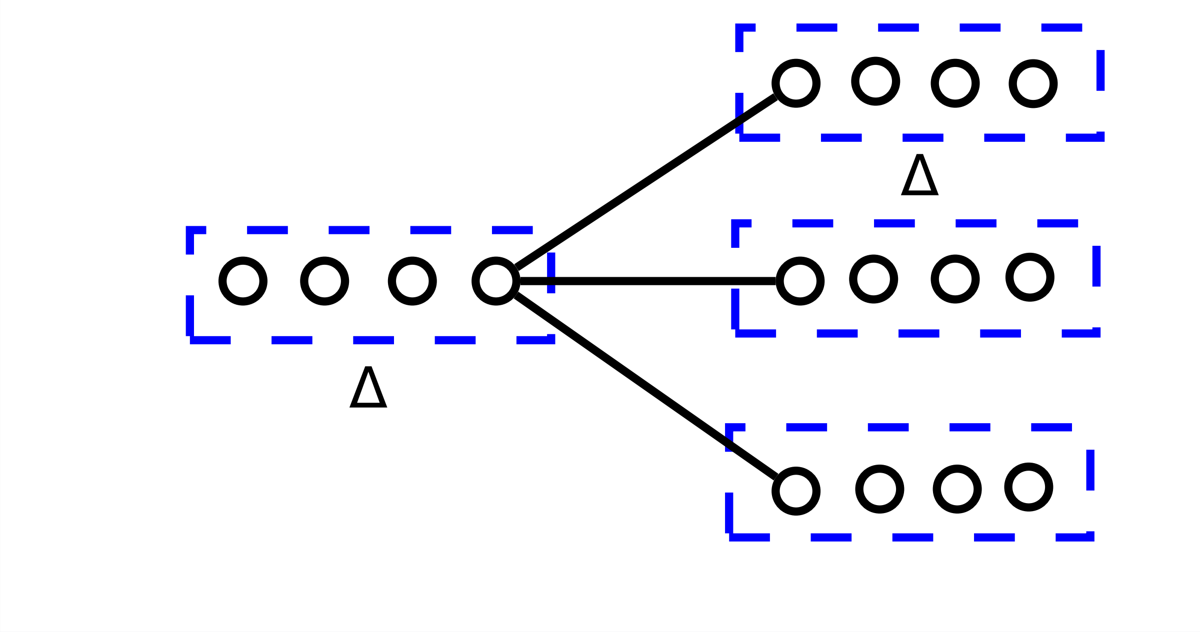
Theorem: Assuming ETH, for all $\delta \in (0,1)$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max Cut with $|E| = n^{1+\delta}$ in time $2^{n^{1-\delta-\epsilon}}$

Starting Point: Max Cut is "APX-ETH"-hard on 5-regular instances with $|V| = n$.



Theorem: Assuming ETH, for all $\delta \in (0,1)$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max Cut with $|E| = n^{1+\delta}$ in time $2^{n^{1-\delta-\epsilon}}$

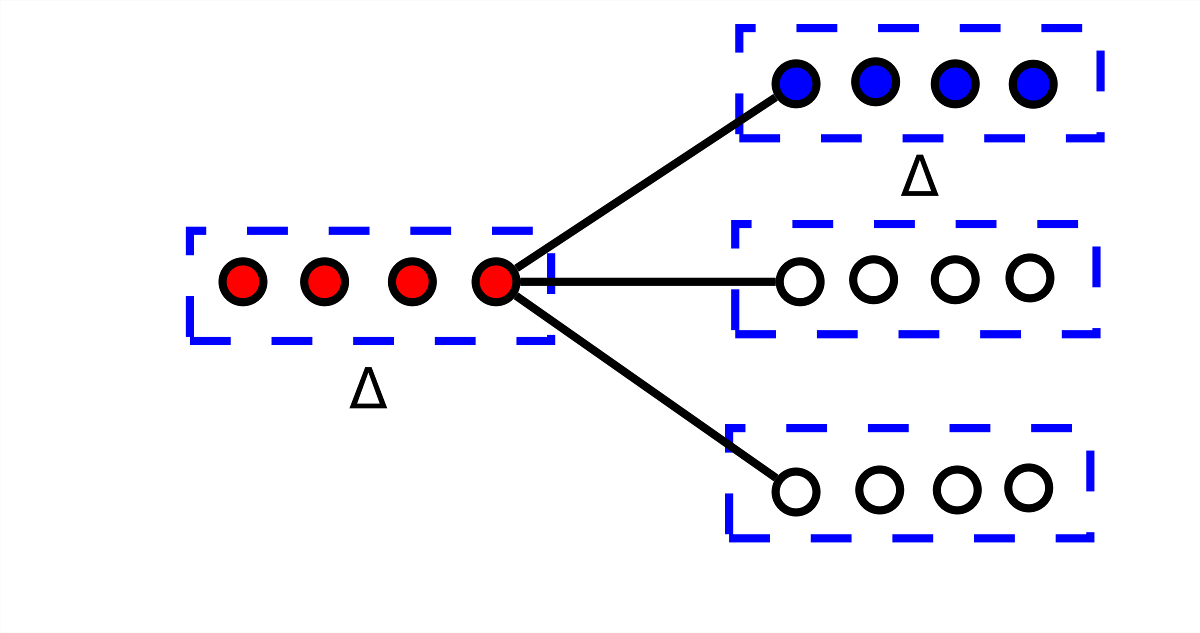
Starting Point: Max Cut is “APX-ETH”-hard on 5-regular instances with $|V| = n$.



Running time lower bound

Theorem: Assuming ETH, for all $\delta \in (0,1)$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max Cut with $|E| = n^{1+\delta}$ in time $2^{n^{1-\delta-\epsilon}}$

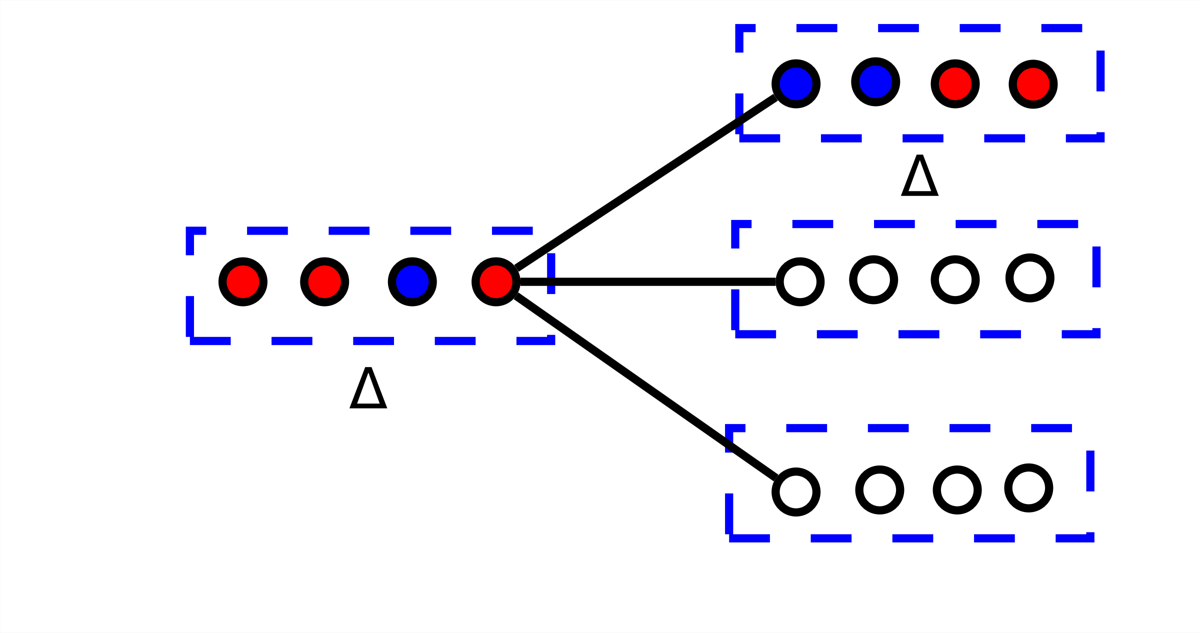
Starting Point: Max Cut is "APX-ETH"-hard on 5-regular instances with $|V| = n$.



Running time lower bound

Theorem: Assuming ETH, for all $\delta \in (0,1)$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max Cut with $|E| = n^{1+\delta}$ in time $2^{n^{1-\delta-\epsilon}}$

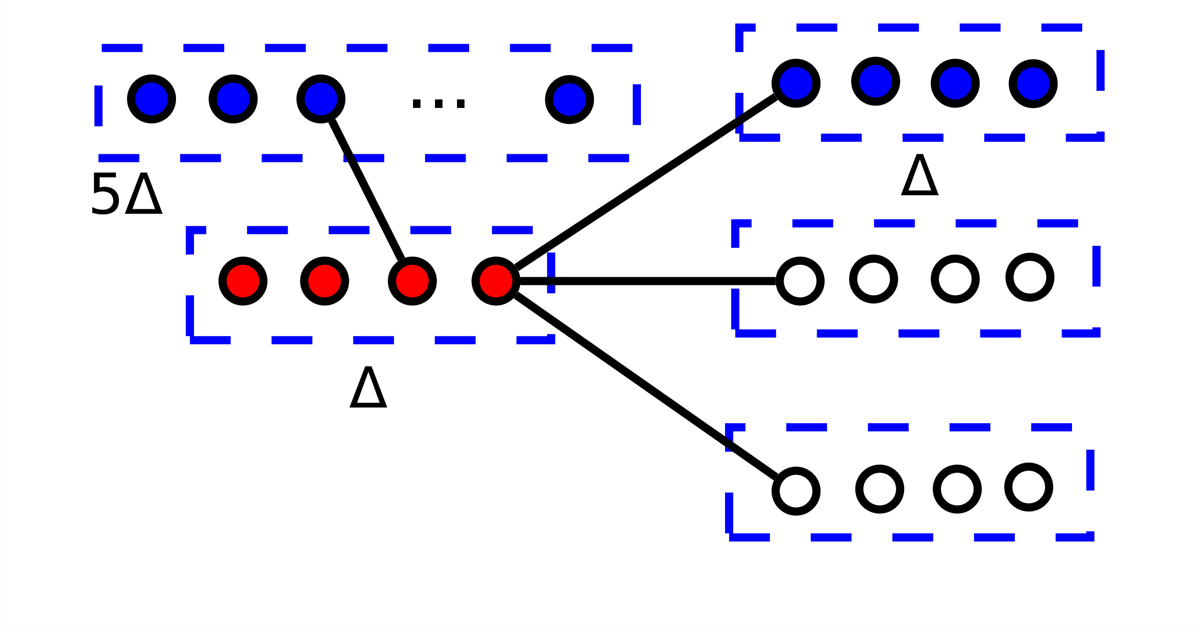
Starting Point: Max Cut is "APX-ETH"-hard on 5-regular instances with $|V| = n$.



Running time lower bound

Theorem: Assuming ETH, for all $\delta \in (0,1)$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max Cut with $|E| = n^{1+\delta}$ in time $2^{n^{1-\delta-\epsilon}}$

Starting Point: Max Cut is "APX-ETH"-hard on 5-regular instances with $|V| = n$.



Running time lower bound

Theorem: Assuming ETH, for all $\delta \in (0,1)$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max Cut with $|E| = n^{1+\delta}$ in time $2^{n^{1-\delta-\epsilon}}$

Starting Point: Max Cut is “APX-ETH”-hard on 5-regular instances with $|V| = n$.

- A constant gap remains for any Δ
- $|V'| = n\Delta$, $|E| = n\Delta^2$, Avg. degree $= \Delta$
- If we cound do better than $2^{|V'|/\Delta}$ then $\neg$ETH

Running time lower bound

Theorem: Assuming ETH, for all $\delta \in (0,1)$, $\exists r < 1$ s.t. $\forall \epsilon > 0$ no algorithm can r-approximate Max Cut with $|E| = n^{1+\delta}$ in time $2^{n^{1-\delta-\epsilon}}$

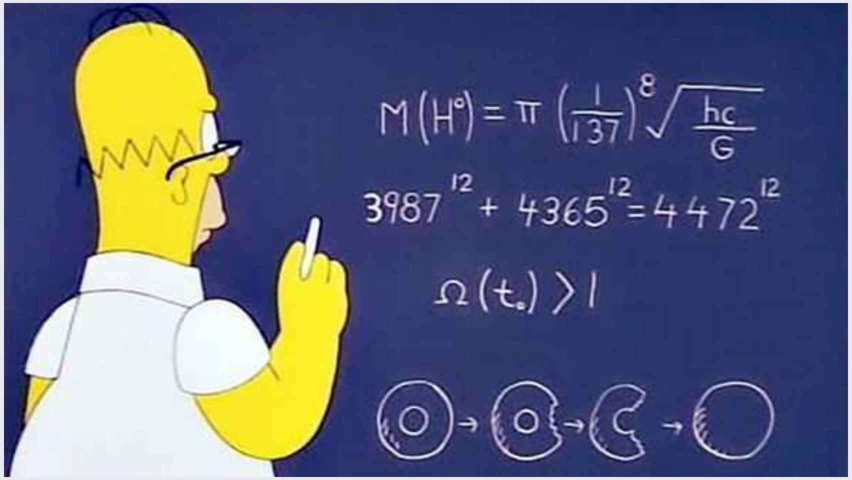
Starting Point: Max Cut is "APX-ETH"-hard on 5-regular instances with $|V| = n$.

- A constant gap remains for any Δ
- $|V'| = n\Delta$, $|E| = n\Delta^2$, Avg. degree $= \Delta$
- If we cound do better than $2^{|V'|/\Delta}$ then $\neg$ETH

- **Bonus:** The two reductions compose! Optimal running times everywhere!

Conclusions

- Density is a crucial parameter for approximating Max-k-CSP
  - Especially useful in sub-exponential setting
  - Smooth trade-off between performance and generality
  - “Tight” bounds
- Lesson: Don’t forget to take into account input structure!



Leveraging Structure to Solve CSP

- Must take into account that input may have some useful properties (otherwise problem too hard!)
- So far, we have used simple properties (density)
- Time to measure structure in a more sophisticated way!
    - **Parameterized Complexity** == Trading Time for Generality
    - Define some distance k from a tractable case (distance from triviality)
    - Try to produce algorithm whose performance **slowly degenerates** as k increases.
- Use tools from Graph Theory to describe input structure.

Structural CSP

- Define some graph structure of ϕ.
- Study CSPs for special graph classes.

Structural CSP

- Define some graph structure of ϕ.

Incidence graph representation of a CSP

- (Unsigned) variables and constraints are represented by vertices;
- a constraint vertex is connected to a variable vertex iff the corresponding constraint involves the corresponding variable.

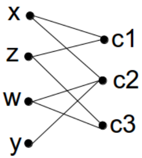


Figure: The incidence graph representation of the previous formula $(\neg x \vee z) \wedge (x \vee y \vee \neg w) \wedge (\neg z \vee w)$.

Structural CSP

- Study CSPs for special graph classes.

Example classes

- Low degree: Bounding degree of incidence graph doesn't help (3CNFSAT where every variable appears at most 3 times is NP-complete).

Example classes

- Low degree: Bounding degree of incidence graph doesn't help (3CNFSAT where every variable appears at most 3 times is NP-complete).

Example classes

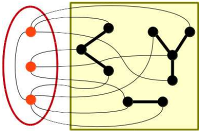
- Low degree: Bounding degree of incidence graph doesn't help (3CNFSAT where every variable appears at most 3 times is NP-complete).
- Acyclicity: Start from the leaves and work your way up (poly-time).

Distance from being acyclic

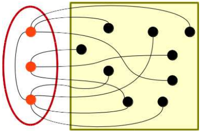
- *Treewidth*

Distance from being acyclic

- *Treewidth*
- *Feedback Vertex Set*: Set of vertices whose removal leaves the graph acyclic.



- *Treewidth*
- *Feedback Vertex Set*: Set of vertices whose removal leaves the graph acyclic.
- *Vertex Cover*: Set of vertices whose removal leaves an independent set.



Distance from being acyclic

- *Treewidth*
- *Feedback Vertex Set*: Set of vertices whose removal leaves the graph acyclic.
- *Vertex Cover*: Set of vertices whose removal leaves an independent set.

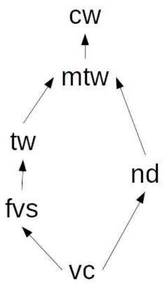
$fvs \leq vc$: Independent set is acyclic.

Distance from being acyclic

- *Treewidth*
- *Feedback Vertex Set*: Set of vertices whose removal leaves the graph acyclic.
- *Vertex Cover*: Set of vertices whose removal leaves an independent set.

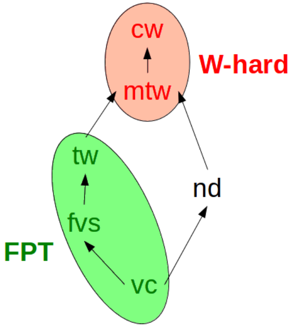
$fvs \leq vc$: Independent set is acyclic.
$tw \leq fvs + 1$:

- Make a tree-decomposition of the forest;
- Put the *fvs* in all bags;

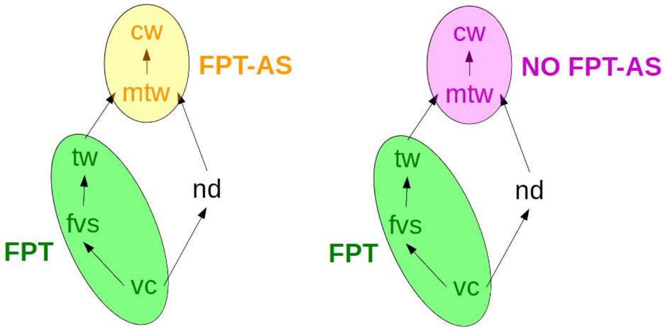


Parameter map: $q \leftarrow p$ (which reads 'q *dominates* p') between two parameters means that q is bounded when p is bounded.

Structural Parameterizations

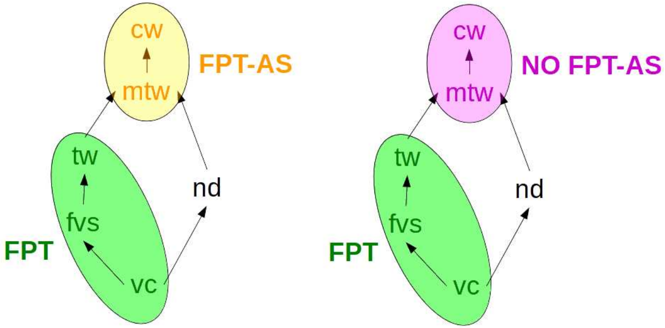


Goal: design algorithms for most dominant parameter (hold downward) and hardness for least dominant (hold upward).



New approach: study FPT approximations to evade hardness. In this talk we examine the existence of FPT Approximation Schemes.

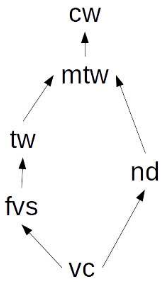
Structural Parameterizations



Definition

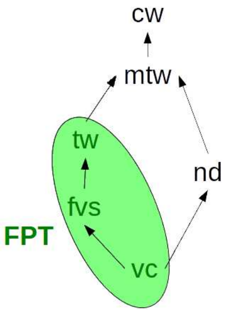
FPT Approximation Scheme (FPT-AS): $\forall \epsilon > 0$ there is an $(1-\epsilon)$-approximation algorithm running in time $O(f(\epsilon, k) \cdot \text{poly}(n))$.

CNFSAT and MAXCNFSAT



Theorem [Szeider 2004]

MAXCNFSAT parameterized by incidence treewidth (tw^*) is FPT.



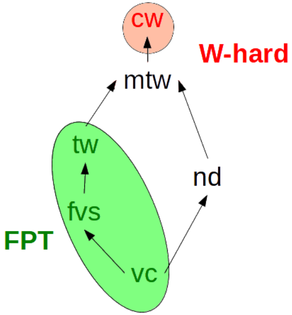
CNFSAT and MAXCNFSAT

Theorem [Szeider 2004]

MAXCNFSAT parameterized by incidence treewidth (tw^*) is FPT.

Theorem [Ordyniak, Paulusma, Szeider 2013]

CNFSAT parameterized by cw^* is W[1]-hard.



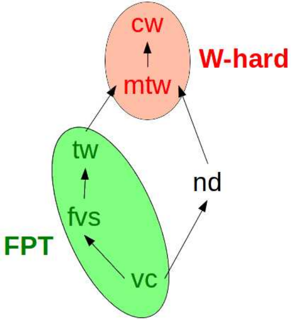
Theorem [Szeider 2004]

MaxCNFSat parameterized by incidence treewidth (tw^*) is FPT.

Theorem [Ordyniak, Paulusma, Szeider 2013]

CNFSat parameterized by cw^* is W[1]-hard.

Hardness even holds for a more restricted parameter *modular treewidth* [Paulusma, Slivovsky, Szeider 2013].



Theorem [Szeider 2004]

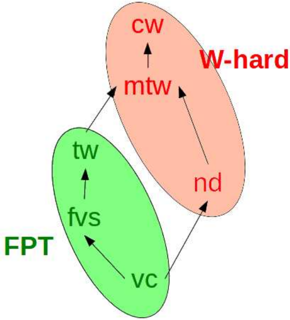
MAXCNFSAT parameterized by incidence treewidth (tw^*) is FPT.

Theorem [Ordyniak, Paulusma, Szeider 2013]

CNFSAT parameterized by cw^* is W[1]-hard.

Hardness even holds for a more restricted parameter *modular treewidth* [Paulusma, Slivovsky, Szeider 2013].

$\rightarrow$ **We extend W[1]-hardness to incidence neighborhood diversity (nd^*).**



Theorem [Szeider 2004]

MAXCNFSAT parameterized by incidence treewidth (tw^*) is FPT.

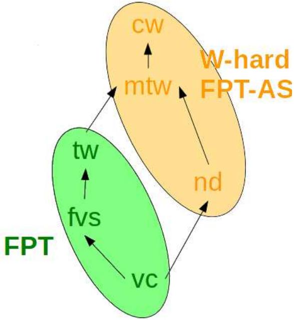
Theorem [Ordyniak, Paulusma, Szeider 2013]

CNFSAT parameterized by cw^* is W[1]-hard.

Hardness even holds for a more restricted parameter *modular treewidth* [Paulusma, Slivovsky, Szeider 2013].

$\rightarrow$ **We extend W[1]-hardness to incidence neighborhood diversity (nd^*).**

$\rightarrow$ **We also present an FPT-AS for cw^*.**



On the positive side...

Theorem

MaxCNFSat *parameterized by* cw^* *admits an FPT-AS.*

On the positive side...

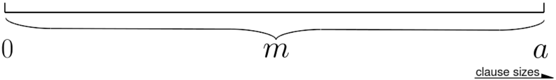
Theorem

MaxCNFSat *parameterized by* cw^* *admits an FPT-AS.*

Reminder

FPT-AS (FPT Approximation Scheme) for a maximization problem parameterized by k: $\forall \epsilon > 0$ there exists an $(1-\epsilon)$-approximation algorithm running in $O(f(\epsilon, k) \cdot \text{poly}(n))$.

An FPT-AS for MaxCNFSat parameterized by cw*

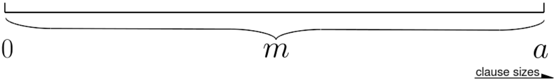


Arrange the clauses in increasing order of arity (0 to a).

An FPT-AS for MAXCNFSAT parameterized by cw*

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

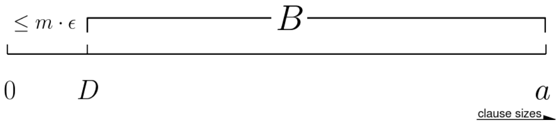
$D = d \cdot \epsilon^4$



Arrange the clauses in increasing order of arity (0 to a).
Split them into big (arity at least $g(\epsilon)$), small (arity at most $g'(\epsilon)$, and medium.
Consider the following cases:

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

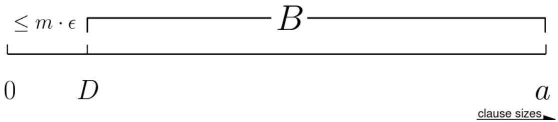
$D = d \cdot \epsilon^4$



(Almost) all clauses are big:

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$

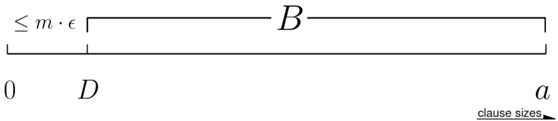


(Almost) all clauses are big:

- ignore small clauses;

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$

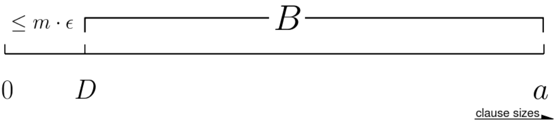


(Almost) all clauses are big:

- ignore small clauses;
- a random assignment satisfies $\geq (1-\epsilon)(1-2^{-g(\epsilon)}) \cdot m$ clauses (with high probability).

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$



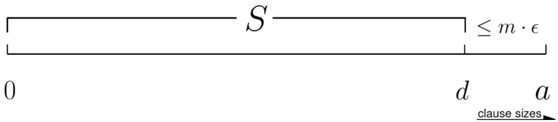
(Almost) all clauses are big:

- ignore small clauses;
- a random assignment satisfies $\geq (1-\epsilon)(1-2^{-g(\epsilon)}) \cdot m$ clauses (with high probability).
- Since $m \geq OPT$, $SOL \geq (1-\epsilon')OPT$, for some ϵ' depending on ϵ.

An FPT-AS for MaxCNFSat parameterized by cw*

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$

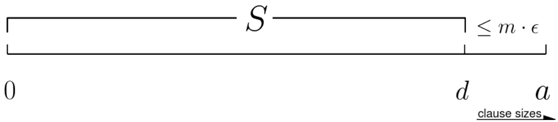


(Almost) all clauses are small:

An FPT-AS for MaxCNFSat parameterized by cw*

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$



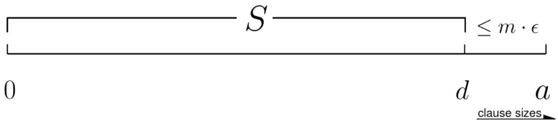
(Almost) all clauses are small:

- ignore large clauses;

An FPT-AS for MaxCNFSat parameterized by cw*

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$



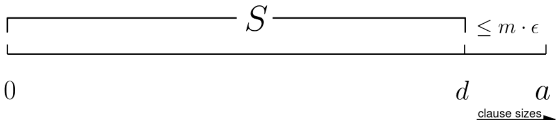
(Almost) all clauses are small:

- ignore large clauses;
- degree on one side of the incidence graph is bounded $\rightarrow$ no large biclique subgraphs;

An FPT-AS for MaxCNFSat parameterized by cw*

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$



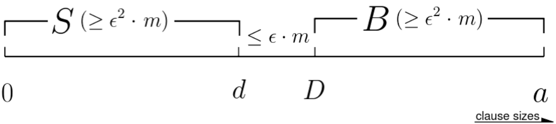
(Almost) all clauses are small:

- ignore large clauses;
- degree on one side of the incidence graph is bounded $\rightarrow$ no large biclique subgraphs;
- By [Gurski, Wanke 2000], the incidence graph has bounded treewidth $\rightarrow$ solve optimally the remaining small clauses;

An FPT-AS for MaxCNFSat parameterized by cw*

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$

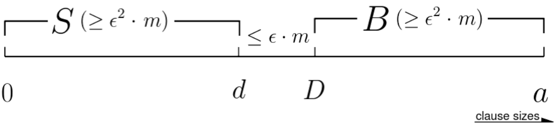


(Almost) no medium-size clauses and B, S are balanced:

An FPT-AS for MaxCNFSat parameterized by cw*

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$



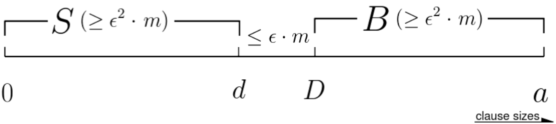
(Almost) no medium-size clauses and B, S are balanced:

- variable occurences$(B) \geq |B| \cdot D = \frac{m \cdot d}{\epsilon^2}$;
- variable occurences$(S) \leq |S| \cdot d \leq m \cdot d$.

An FPT-AS for MaxCNFSat parameterized by cw^*

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$



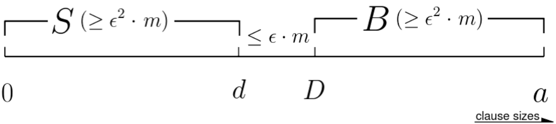
(Almost) no medium-size clauses and B, S are balanced:

- variable occurences$(B) \geq |B| \cdot D = \frac{m \cdot d}{\epsilon^2}$;
- variable occurences$(S) \leq |S| \cdot d \leq m \cdot d$.

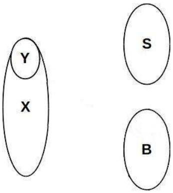
$\rightarrow \exists y \in V$ that appears $1/\epsilon^2$ more times in B than in S.

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$



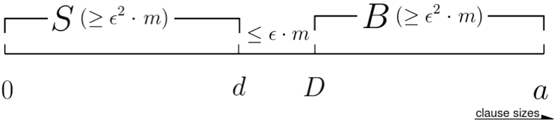
(Almost) no medium-size clauses and B, S are balanced:
From the previous observation, we iteratively create a set of variables Y with the following properties:



An FPT-AS for MaxCNFSat parameterized by cw*

B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

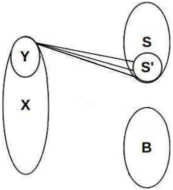
$D = d \cdot \epsilon^4$



(Almost) no medium-size clauses and B, S are balanced:

From the previous observation, we iteratively create a set of variables Y with the following properties:

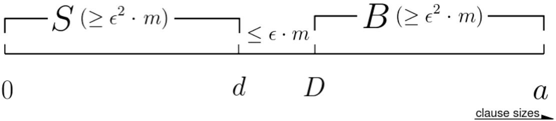
- Y *hits* few clauses of S (call this set S');



An FPT-AS for MaxCNFSat parameterized by cw*

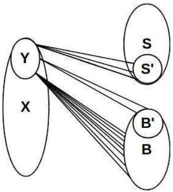
B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$



(Almost) no medium-size clauses and B, S are balanced:
From the previous observation, we iteratively create a set of variables Y with the following properties:

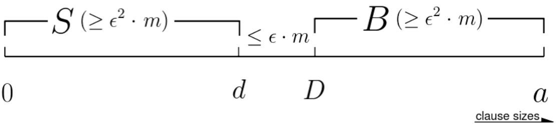
- Y *hits* few clauses of S (call this set S');
- at most ϵ^2 clauses of B have $\leq 1/\epsilon$ neighbors in Y (call this set B').



An FPT-AS for MaxCNFSat parameterized by cw*

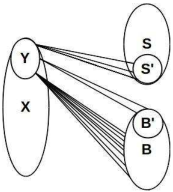
B: Clauses of arity $\geq D = g(\epsilon)$.
S: Clauses of arity $\leq d = g'(\epsilon)$.

$D = d \cdot \epsilon^4$



(Almost) no medium-size clauses and B, S are balanced:
From the previous observation, we iteratively create a set of variables Y with the following properties:

- Y *hits* few clauses of S (call this set S');
- at most ϵ^2 clauses of B have $\leq 1/\epsilon$ neighbors in Y (call this set B').



Randomly assigning Y should satisfy whp $\geq (1-\epsilon^2)\cdot(1-2^{-1/\epsilon})$ of $B \setminus B'$, while $S \setminus S'$ can be solved optimally.

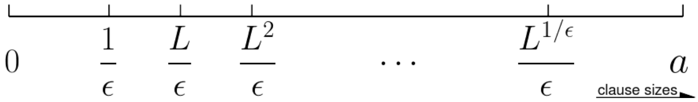
An FPT-AS for MaxCNFSat parameterized by cw^*

Lemma

We can always find a small set M ($|M| \leq \epsilon \cdot m$) of medium-size clauses (arities $d \sim D$).

Lemma

We can always find a small set M ($|M| \leq \epsilon \cdot m$) of medium-size clauses (arities $d \sim D$).

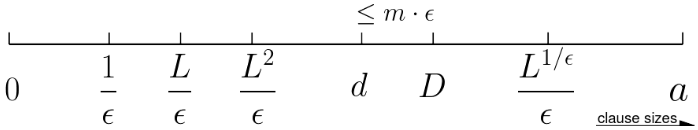


Define $1/\epsilon + 1$ independent intervals of medium-arity clauses (right-left bounds are an $L(= \epsilon^{-4})$-factor apart).

An FPT-AS for MAXCNFSAT parameterized by cw*

Lemma

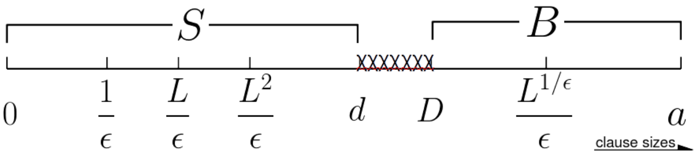
We can always find a small set M ($|M| \leq \epsilon \cdot m$) of medium-size clauses (arities $d \sim D$).



There should be at least one interval $[d, D]$ ($D = L \cdot d$) containing $\leq \epsilon \cdot m$ clauses.

Lemma

We can always find a small set M ($|M| \leq \epsilon \cdot m$) of medium-size clauses (arities $d \sim D$).



Removing them divides the clauses into small (S) and big (B).

The algorithm

- Find interval [d,D] of at most $\epsilon \cdot m$ clauses of medium arities as in the previous Lemma and ignore them.

The algorithm

- Find interval [d,D] of at most $\epsilon \cdot m$ clauses of medium arities as in the previous Lemma and ignore them.
- Split remaining clauses into S (arity $< d$) and B (arity $> D$).

The algorithm

- Find interval [d,D] of at most $\epsilon \cdot m$ clauses of medium arities as in the previous Lemma and ignore them.
- Split remaining clauses into S (arity $< d$) and B (arity $> D$).
- If $|S| \leq \epsilon^2 \cdot m$

The algorithm

- Find interval [d,D] of at most $\epsilon \cdot m$ clauses of medium arities as in the previous Lemma and ignore them.
- Split remaining clauses into S (arity $< d$) and B (arity $> D$).
- If $|S| \leq \epsilon^2 \cdot m$
  - Ignore S;

The algorithm

- Find interval [d,D] of at most $\epsilon \cdot m$ clauses of medium arities as in the previous Lemma and ignore them.
- Split remaining clauses into S (arity $< d$) and B (arity $> D$).
- If $|S| \leq \epsilon^2 \cdot m$
  - Ignore S;
  - Randomly assign variables to satisfy most of B.

The algorithm

- Find interval [d,D] of at most $\epsilon \cdot m$ clauses of medium arities as in the previous Lemma and ignore them.
- Split remaining clauses into S (arity $< d$) and B (arity $> D$).
- If $|S| \leq \epsilon^2 \cdot m$
  - Ignore S;
  - Randomly assign variables to satisfy most of B.
- If at most $|B| \leq \epsilon^2 \cdot m$

The algorithm

- Find interval [d,D] of at most $\epsilon \cdot m$ clauses of medium arities as in the previous Lemma and ignore them.
- Split remaining clauses into S (arity $< d$) and B (arity $> D$).
- If $|S| \leq \epsilon^2 \cdot m$
  - Ignore S;
  - Randomly assign variables to satisfy most of B.
- If at most $|B| \leq \epsilon^2 \cdot m$
  - Ignore B;

The algorithm

- Find interval [d,D] of at most $\epsilon \cdot m$ clauses of medium arities as in the previous Lemma and ignore them.
- Split remaining clauses into S (arity $< d$) and B (arity $> D$).
- If $|S| \leq \epsilon^2 \cdot m$
  - Ignore S;
  - Randomly assign variables to satisfy most of B.
- If at most $|B| \leq \epsilon^2 \cdot m$
  - Ignore B;
  - G_S^* has bounded treewidth $\rightarrow$ solve optimally.

The algorithm

- Find interval [d,D] of at most $\epsilon \cdot m$ clauses of medium arities as in the previous Lemma and ignore them.
- Split remaining clauses into S (arity $< d$) and B (arity $> D$).
- If $|S| \leq \epsilon^2 \cdot m$
  - Ignore S;
  - Randomly assign variables to satisfy most of B.
- If at most $|B| \leq \epsilon^2 \cdot m$
  - Ignore B;
  - G_S^* has bounded treewidth $\rightarrow$ solve optimally.
- Otherwise

The algorithm

- Find interval [d,D] of at most $\epsilon \cdot m$ clauses of medium arities as in the previous Lemma and ignore them.
- Split remaining clauses into S (arity $< d$) and B (arity $> D$).
- If $|S| \leq \epsilon^2 \cdot m$
  - Ignore S;
  - Randomly assign variables to satisfy most of B.
- If at most $|B| \leq \epsilon^2 \cdot m$
  - Ignore B;
  - G_S^* has bounded treewidth $\rightarrow$ solve optimally.
- Otherwise
  - Find set of variables Y as in the last case and set it randomly to satisfy most of B.

The algorithm

- Find interval [d,D] of at most $\epsilon \cdot m$ clauses of medium arities as in the previous Lemma and ignore them.
- Split remaining clauses into S (arity $< d$) and B (arity $> D$).
- If $|S| \leq \epsilon^2 \cdot m$
  - Ignore S;
  - Randomly assign variables to satisfy most of B.
- If at most $|B| \leq \epsilon^2 \cdot m$
  - Ignore B;
  - G_S^* has bounded treewidth $\rightarrow$ solve optimally.
- Otherwise
  - Find set of variables Y as in the last case and set it randomly to satisfy most of B.
  - Ignore part of S that contains variables from Y and solve the rest optimally.

Conclusions

- Trading **Time-Generality-Approximation**
- Crucial: Take into account input structure!
- Long-term Goal: Map out complete trade-offs
  - For each desired approximation ratio, for each class of inputs (defined by k), what is the correct running time?

A concrete problem for ESIGMA

- Use these techniques for **approximate formula representation** (aka formula learning/knowledge compilation).
- What are the key measures of input structure?

Thank you!
Questions?