Trading Time for Approximation

Michael Lampis

Université Paris Dauphine

MTA SZTAKI – Sep 21 2016

Motivation – Sub-Exponential Approximation

(50 years in a slide)

Motivation – Sub-Exponential Approximation

(50 years in a slide)

- Efficient = Poly-time ('60s)

Jack Edmonds

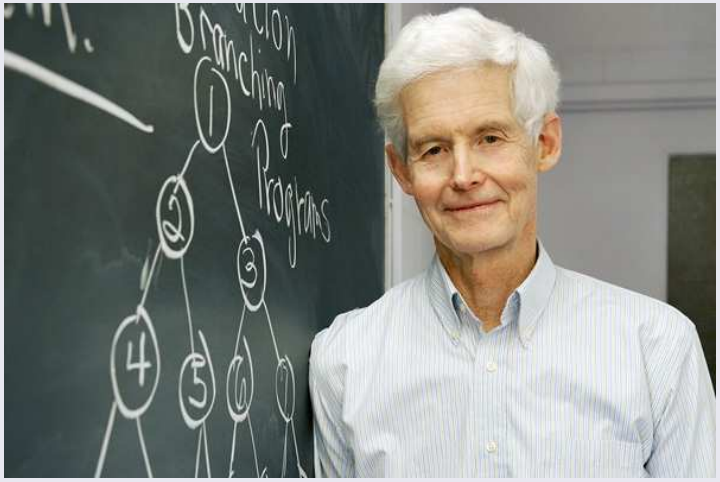
Juris Hartmanis

Richard Stearns

Motivation – Sub-Exponential Approximation

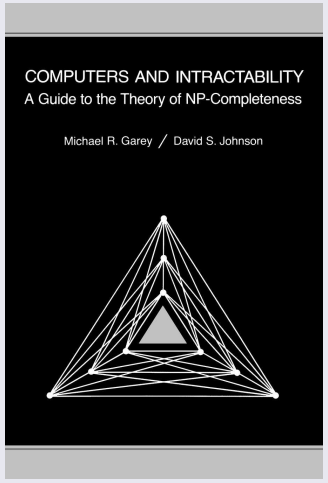
(50 years in a slide)

- Efficient = Poly-time ('60s)
- Everything is NP-hard!(*) ('70s)



Stephen Cook

Richard Karp



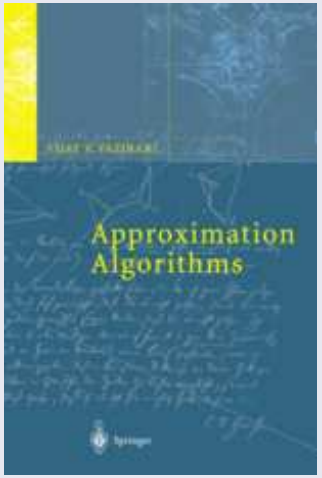
Garey& Johnson

Motivation – Sub-Exponential Approximation

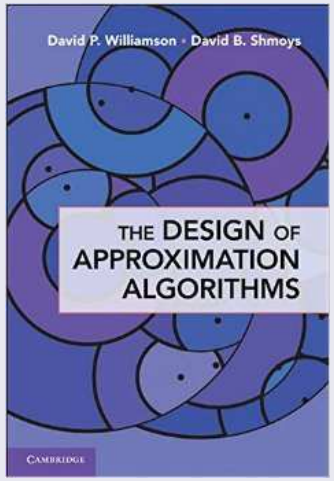
(50 years in a slide)

- Efficient = Poly-time ('60s)
- Everything is NP-hard!(*) ('70s)
- So, we should approximate ('80s)

David Johnson



V. Vazirani

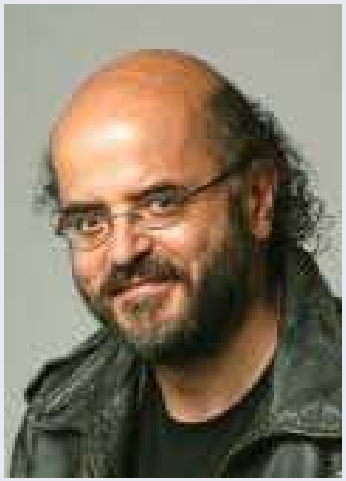


Williamson&Shmoys

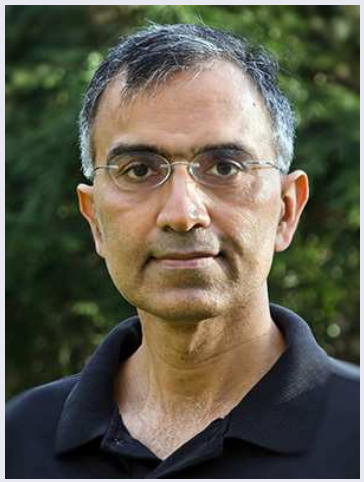
Motivation – Sub-Exponential Approximation

(50 years in a slide)

- Efficient = Poly-time ('60s)
- Everything is NP-hard!(*) ('70s)
- So, we should approximate ('80s)
- Everything is APX-hard!(*) ('90s)



Christos Papadimitriou



Sanjeev Arora

Johan Håstad

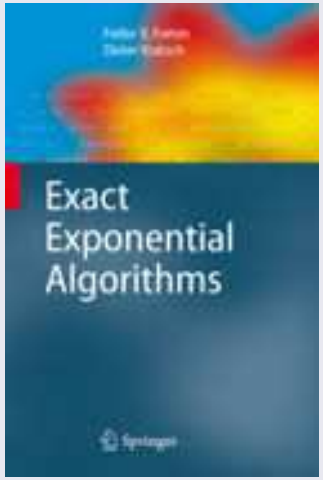
Motivation – Sub-Exponential Approximation

(50 years in a slide)

- Efficient = Poly-time ('60s)
- Everything is NP-hard!(*) ('70s)
- So, we should approximate ('80s)
- Everything is APX-hard!(*) ('90s)
- More than poly-time? Everything ETH-hard ('00s)

Mike Fellows

Russell Impagliazzo



Fomin&Kratsch

Motivation – Sub-Exponential Approximation

(50 years in a slide)

- Efficient = Poly-time ('60s)
- Everything is NP-hard!(*) ('70s)
- So, we should approximate ('80s)
- Everything is APX-hard!(*) ('90s)
- More than poly-time? Everything ETH-hard ('00s)

Bottom line:

- Most problems hard to solve exactly in $2^{o(n)}$ time
- Most problems hard to approximate in $n^{O(1)}$ time

- $\rightarrow$ Perhaps we can approximate in $2^{o(n)}$ time?
- More broadly:
  - Does a **little extra time** give **much better approximation**?
  - Is a **good approximation** much faster than **solving exactly**?

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?

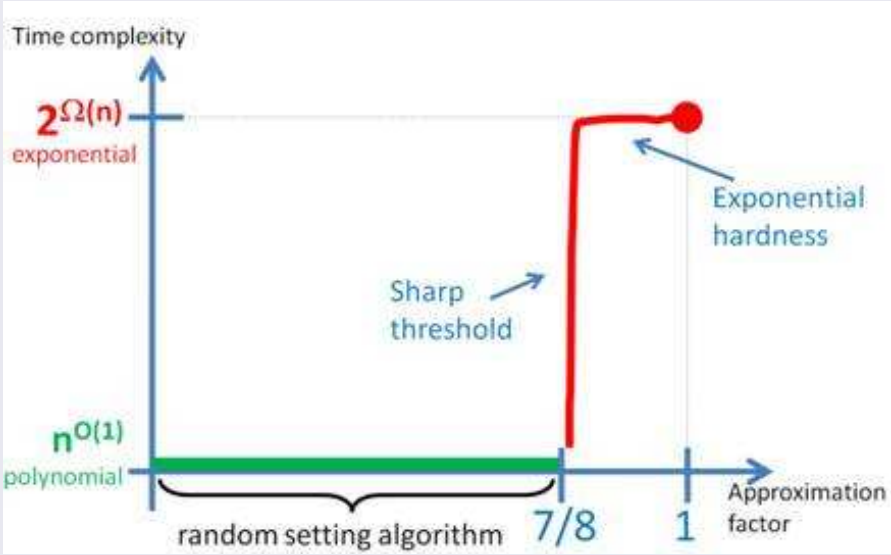
- Better than $3/2$ for TSP, $4/3$ for Max-3-DM, $7/8$ for Max-3-SAT,. . . , in **sub-exponential time**?

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 about problems outside APX?
    - Independent Set?
    - Set Cover?
  - **What else** can we use the extra time for? (if anything)

Approximations for Hard Problems

Very Hard Problems

Joint work with:

Edouard Bonnet Vangelis Paschos

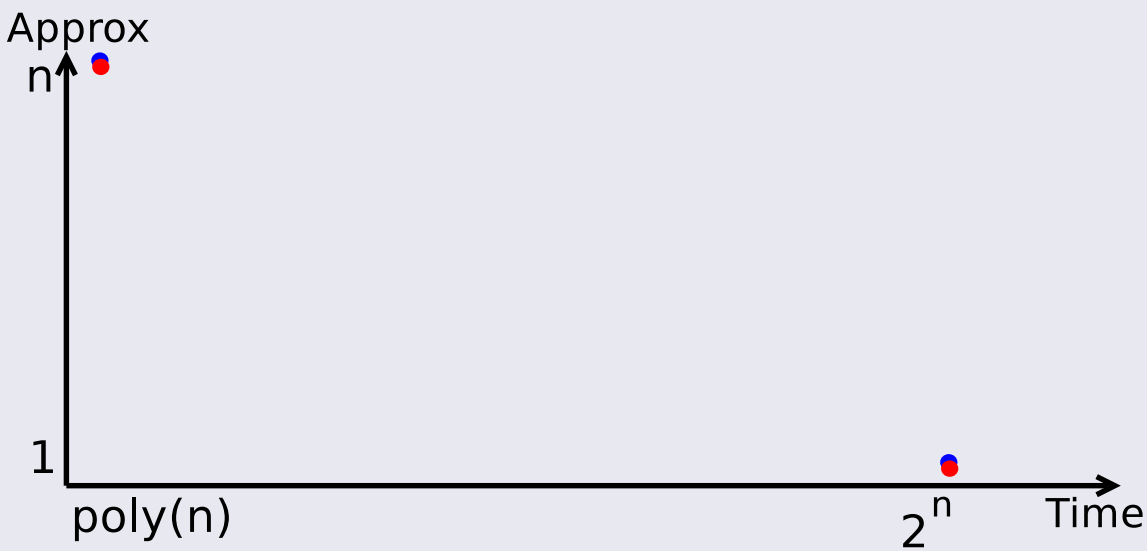
"Time-Approximation Trade-Offs for Inapproximable Problems", STACS '16

Very Hard Problems

- Consider problems which are really hard to approximate in polynomial time.
- Does extra time at least help with these?

Very Hard Problems

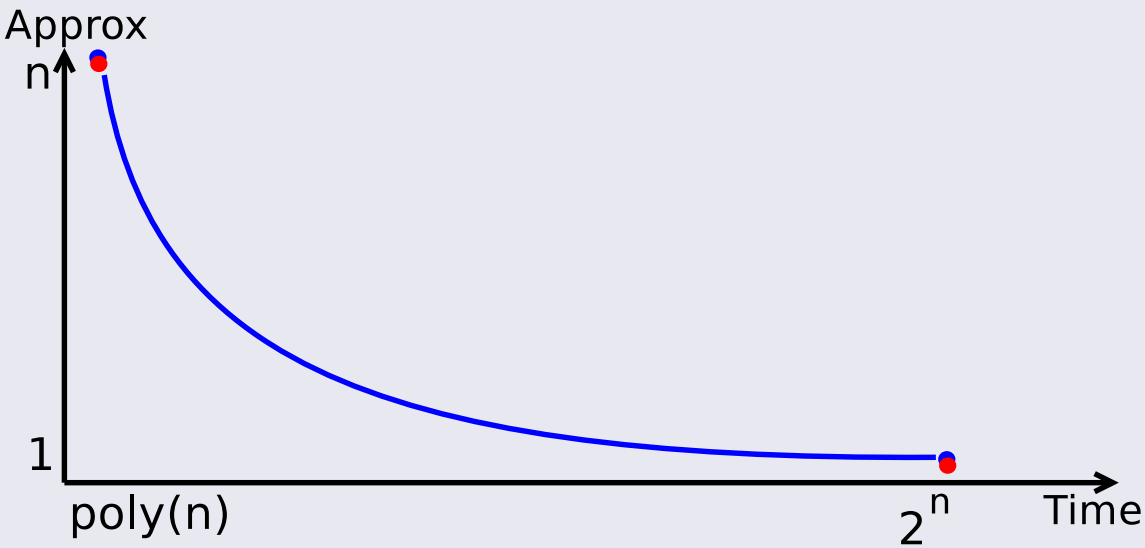
- Consider problems which are really hard to approximate in polynomial time.
- Does extra time at least help with these?
- Example: Independent Set



- Approximation ratio in poly time $n^{1-\epsilon}$
- No $2^{o(n)}$ exact algorithm

Very Hard Problems

- Consider problems which are really hard to approximate in polynomial time.
- Does extra time at least help with these?
- Example: Independent Set



- Simple r-approximation in $2^{n/r}$

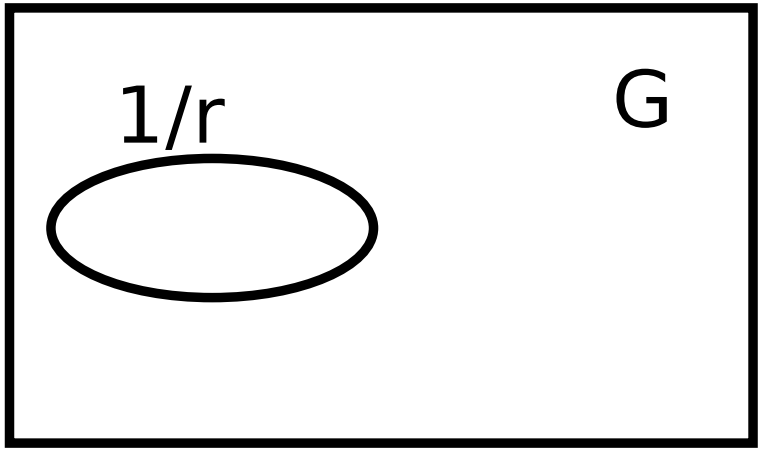
Very Hard Problems

- Consider problems which are really hard to approximate in polynomial time.
- Does extra time at least help with these?
- Example: Independent Set

Randomly sample n/r vertices, find their Max IS

Very Hard Problems

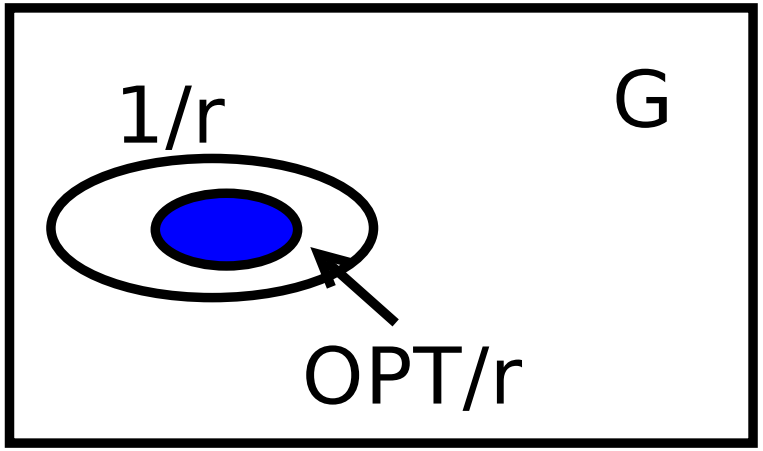
- Consider problems which are really hard to approximate in polynomial time.
- Does extra time at least help with these?
- Example: Independent Set



Randomly sample n/r vertices, find their Max IS

Very Hard Problems

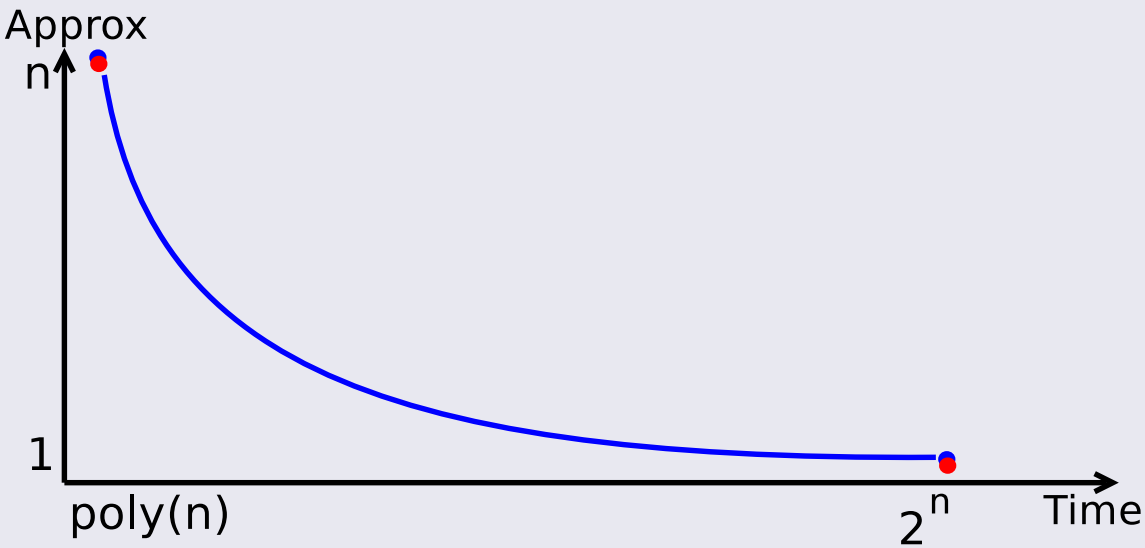
- Consider problems which are really hard to approximate in polynomial time.
- Does extra time at least help with these?
- Example: Independent Set



Randomly sample n/r vertices, find their Max IS

Very Hard Problems

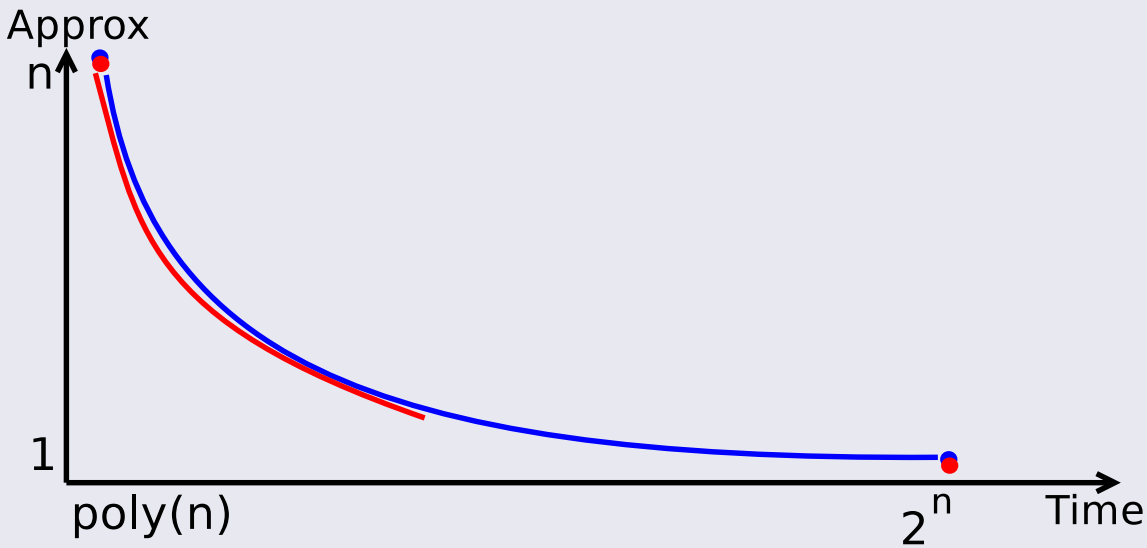
- Consider problems which are really hard to approximate in polynomial time.
- Does extra time at least help with these?
- Example: Independent Set



- Simple r-approximation in $2^{n/r}$

Very Hard Problems

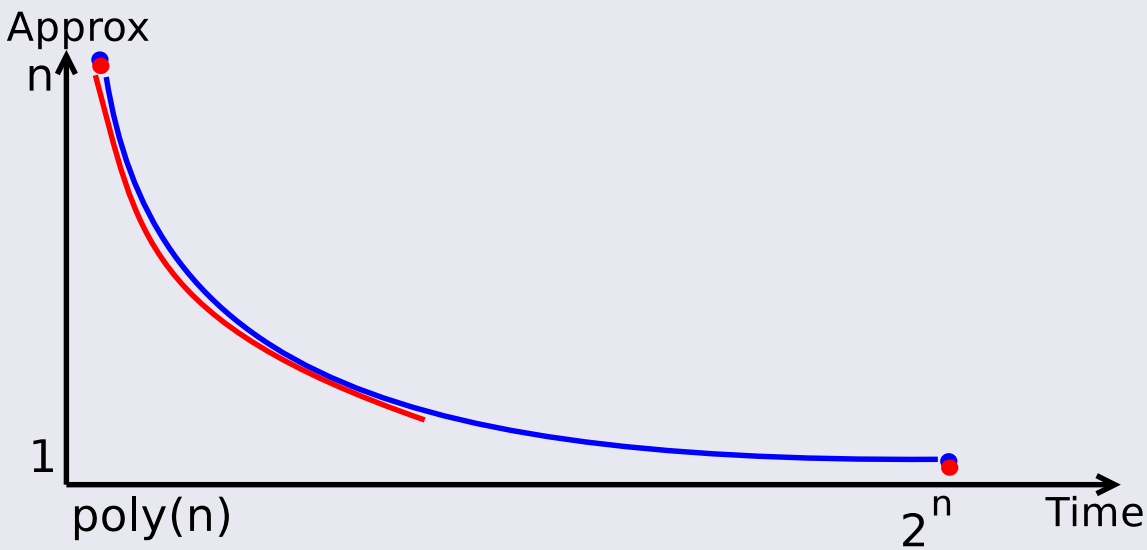
- Consider problems which are really hard to approximate in polynomial time.
- Does extra time at least help with these?
- Example: Independent Set



- For any $0 < r < \sqrt{n}$ there is no r-approximation for IS in time $2^{(n/r)^{1-\epsilon}}$. [Chalermsook, Laekhanukit, Nanongkai] (FOCS'11)

Very Hard Problems

- Consider problems which are really hard to approximate in polynomial time.
- Does extra time at least help with these?
- Example: Independent Set



- Proof idea: Reduce 3-SAT with n vars to IS with rn vertices.
- Either $\mathrm{OPT} \approx rn$ or $\mathrm{OPT} \approx n$.

Very Hard Problems

- Consider problems which are really hard to approximate in polynomial time.
- Does extra time at least help with these?
- Example: Independent Set

Can we **completely** determine the trade-off curve for other problems?

- How much time do we need for r-approximation?
- What is the best approximation in time T?

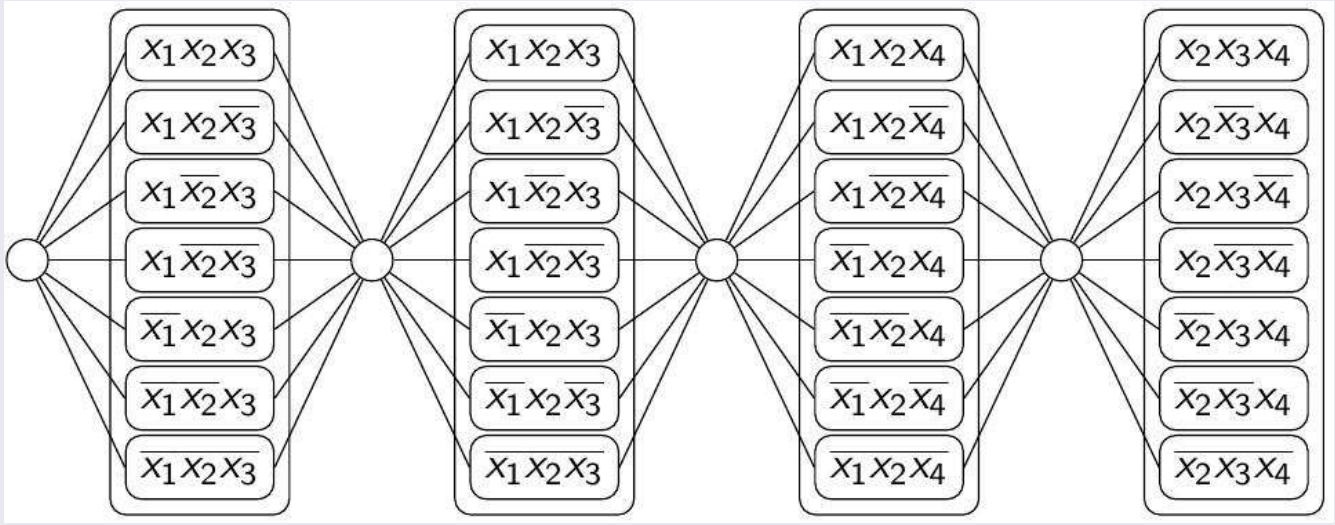
Maximum Induced Path

Theorem: For any parameter $r > 0$, there is no r-approximation for Max Induced Path running in time $2^{o(n/r)}$.

Problem: Find maximum number of vertices that **induce** a path

Maximum Induced Path

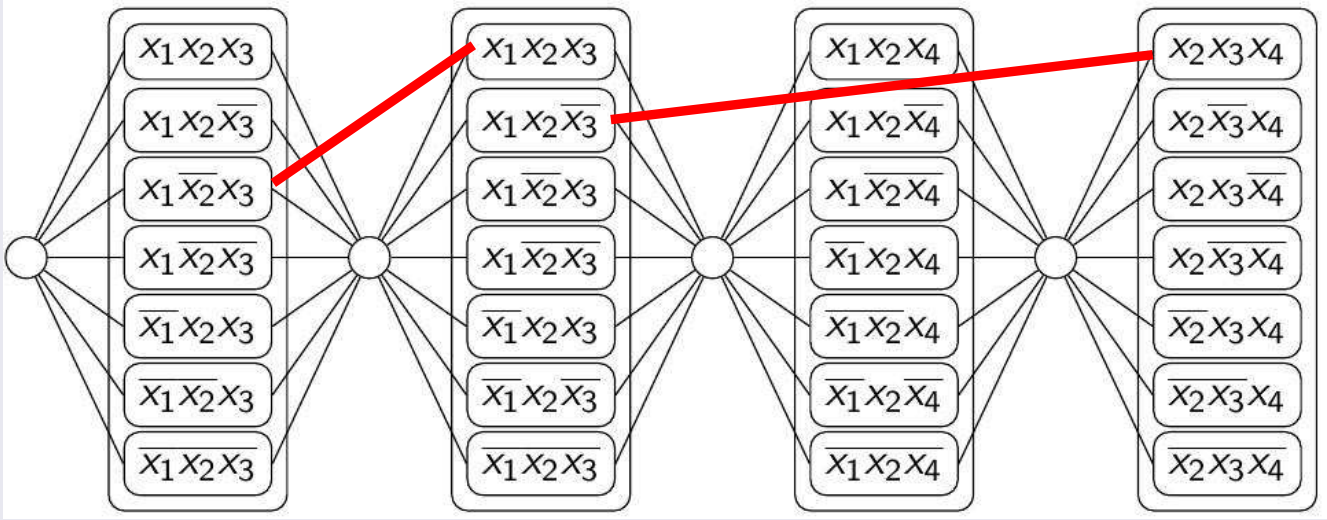
Theorem: For any parameter $r > 0$, there is no r-approximation for Max Induced Path running in time $2^{o(n/r)}$.



Reduction from 3-SAT

Maximum Induced Path

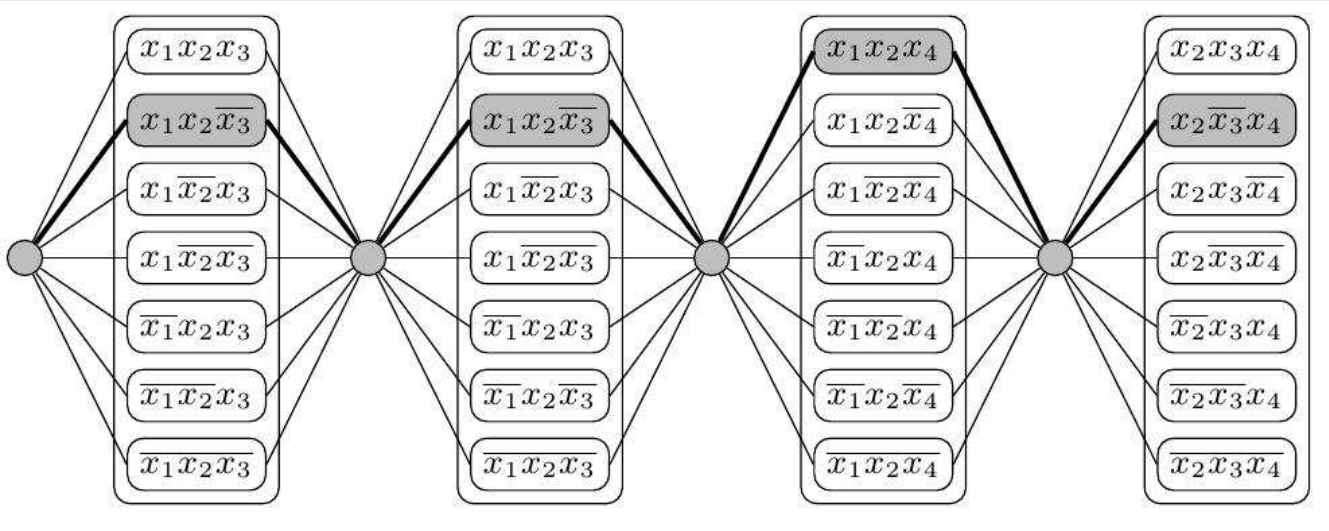
Theorem: For any parameter $r > 0$, there is no r-approximation for Max Induced Path running in time $2^{o(n/r)}$.



Reduction from 3-SAT
One node for every clause assignment
Extra edges for inconsistent assignments

Maximum Induced Path

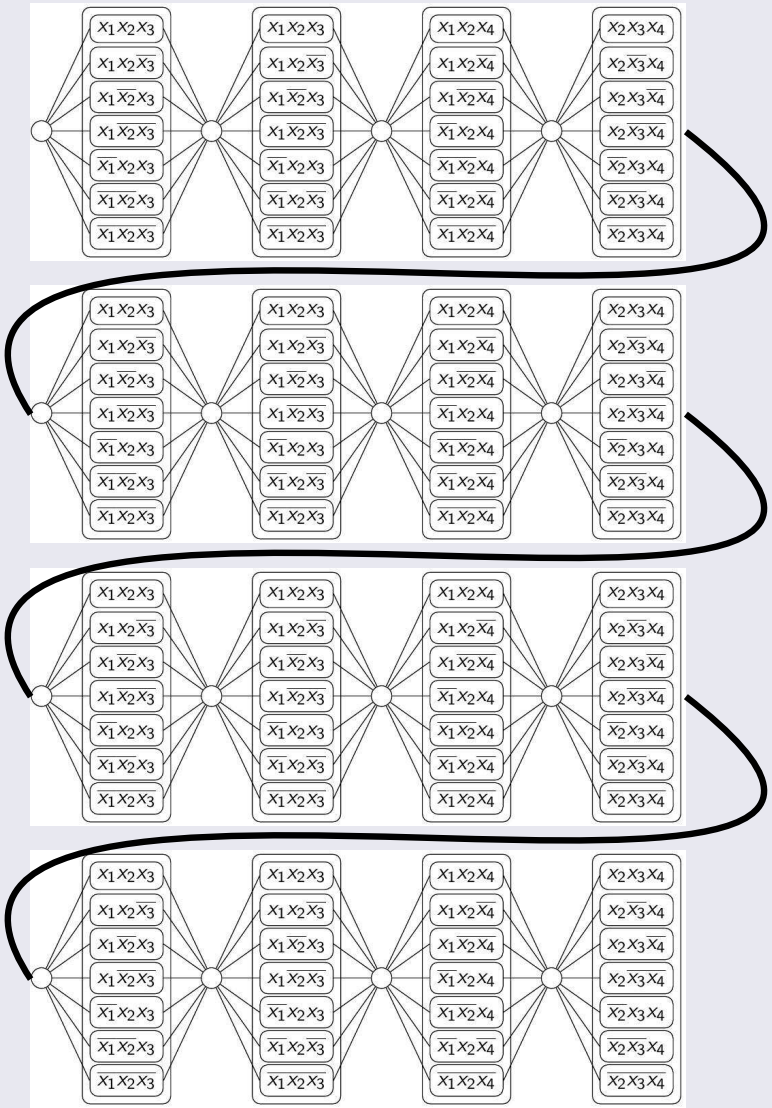
Theorem: For any parameter $r > 0$, there is no r-approximation for Max Induced Path running in time $2^{o(n/r)}$.



Reduction from 3-SAT
One node for every clause assignment
Extra edges for inconsistent assignments
Consistent Global Assignment $\rightarrow$ Induced Path

Maximum Induced Path

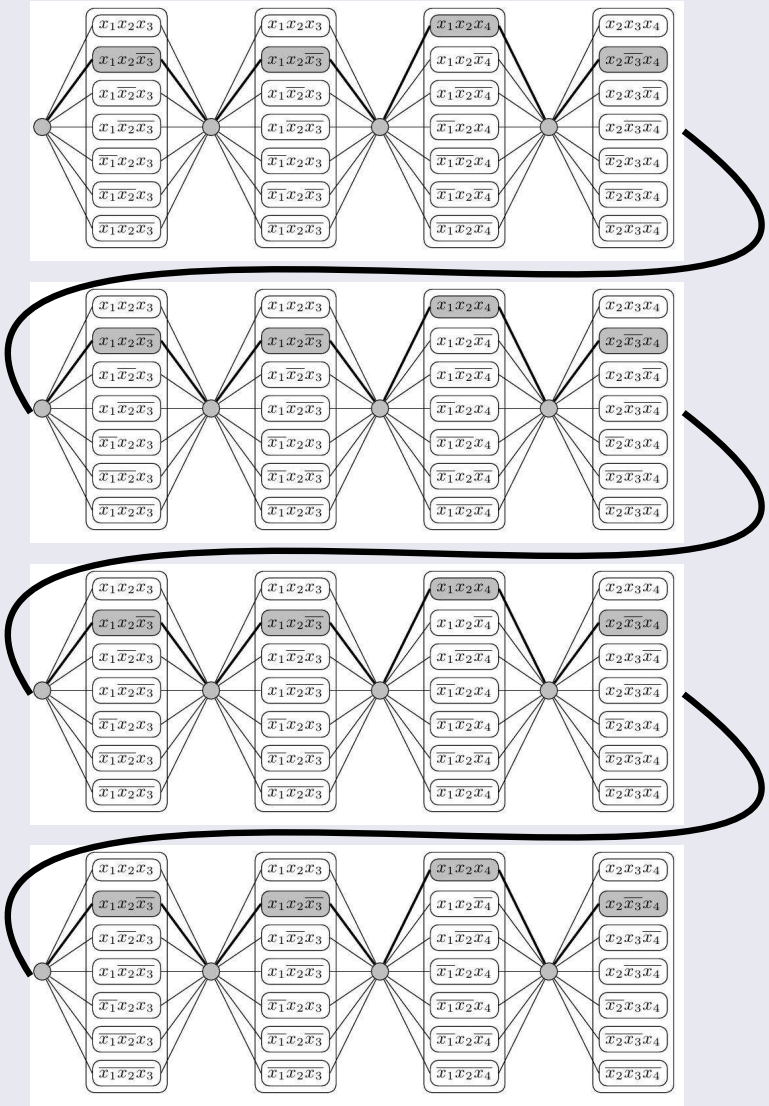
Theorem: For any parameter $r > 0$, there is no r-approximation for Max Induced Path running in time $2^{o(n/r)}$.



Take r copies

Maximum Induced Path

Theorem: For any parameter $r > 0$, there is no r-approximation for Max Induced Path running in time $2^{o(n/r)}$.



Take r copies
Assignment $\rightarrow$ Induced Path
Size = rn
Otherwise $\leq n$
Graph size = $O(rn)$

Maximum Induced Path

Theorem: For any parameter $r > 0$, there is no r-approximation for Max Induced Path running in time $2^{o(n/r)}$.

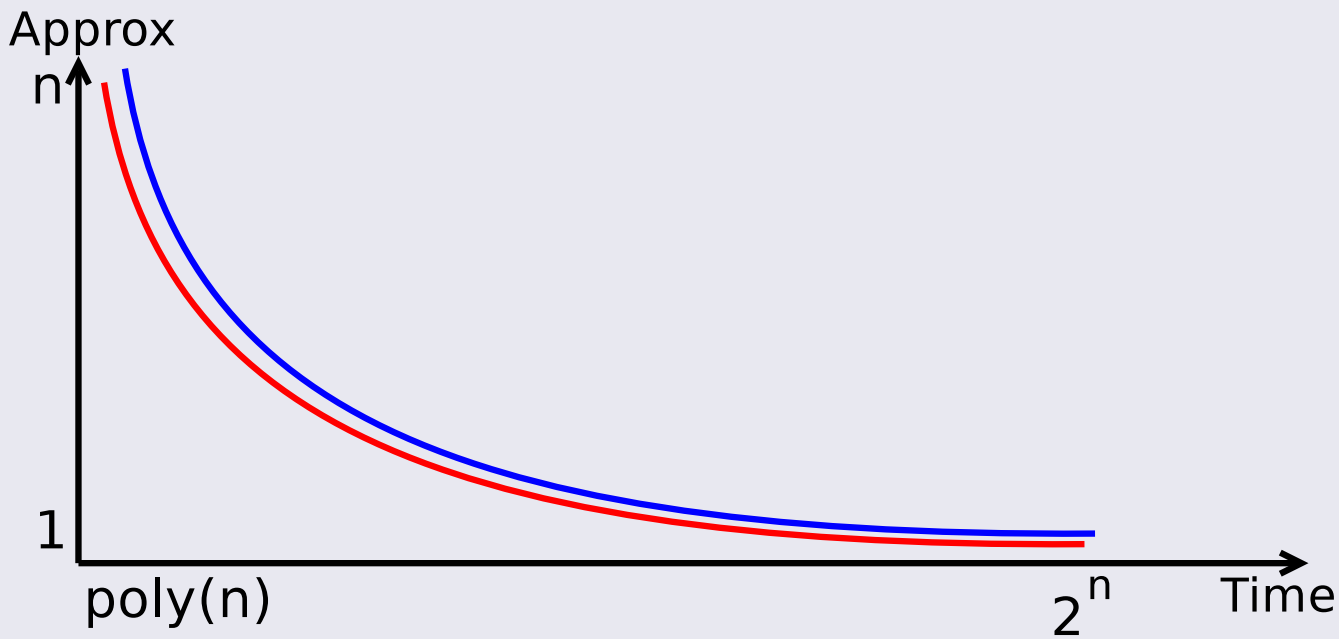
Theorem: For any parameter $r > 0$, there is an r-approximation for Max Induced Path running in time $r^{O(n/r)}$.

Proof: Try out all subsets of size $\leq r$.

Maximum Induced Path

Theorem: For any parameter $r > 0$, there is no r-approximation for Max Induced Path running in time $2^{o(n/r)}$.

Theorem: For any parameter $r > 0$, there is an r-approximation for Max Induced Path running in time $r^{O(n/r)}$.



Can we make these match?

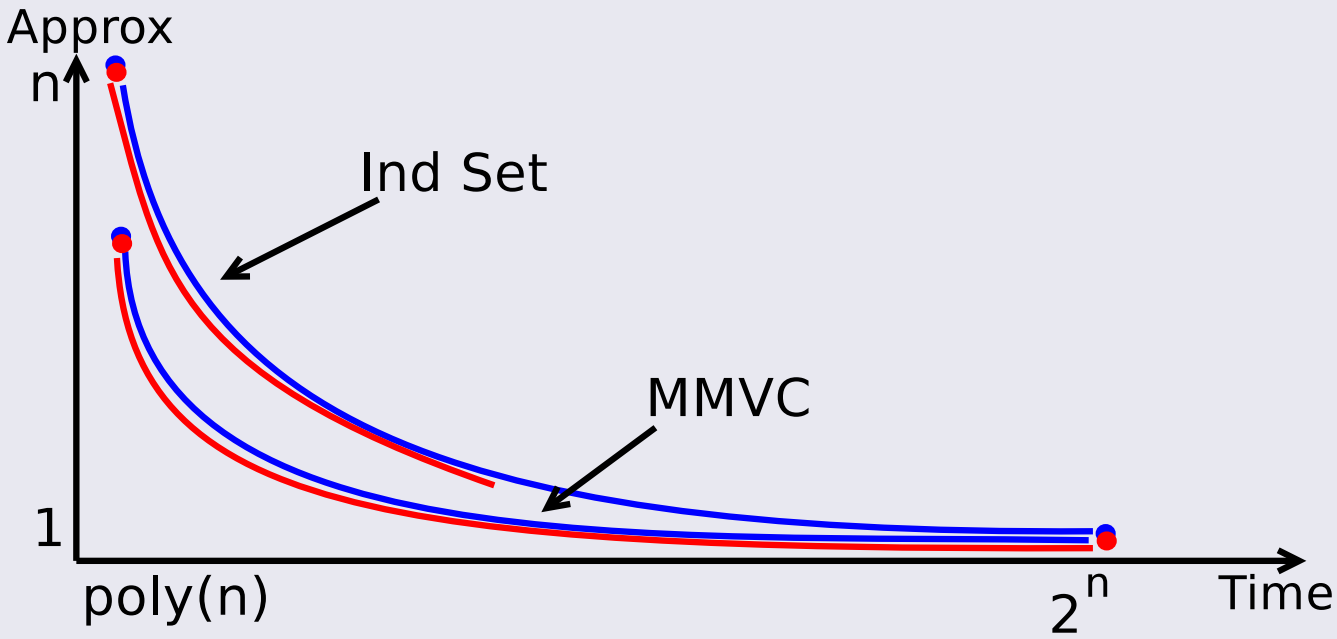
Maximum Minimal Vertex Cover

Theorem: For any parameter $r > 0$, there is no r-approximation for Max Minimal VC running in time $2^{(n/r^2)^{1-\epsilon}}$.
Theorem: For any parameter $r > 0$, there is an r-approximation for Max Minimal VC running in time $2^{O(n/r^2)}$.

Maximum Minimal Vertex Cover

Theorem: For any parameter $r > 0$, there is no r-approximation for Max Minimal VC running in time $2^{(n/r^2)^{1-\epsilon}}$.

Theorem: For any parameter $r > 0$, there is an r-approximation for Max Minimal VC running in time $2^{O(n/r^2)}$.



Summary for hard problems

- Extra time can buy us much better approximation!
  - But only for really hard problems.
- We can completely determine the optimal trade-off!
  - But only for really hard problems.
- Can the extra time buy us anything else?

Trading Time for Generality

An island of Tractability

Joint work with:

Dimitris Fotakis

Vangelis Paschos

"Sub-Exponential Approximation Schemes for CSPs: from Dense to Almost-Sparse", STACS '16

An island of Tractability

- 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), (de la Vega '96)

An island of Tractability

- 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), (de la Vega '96)

An island of Tractability

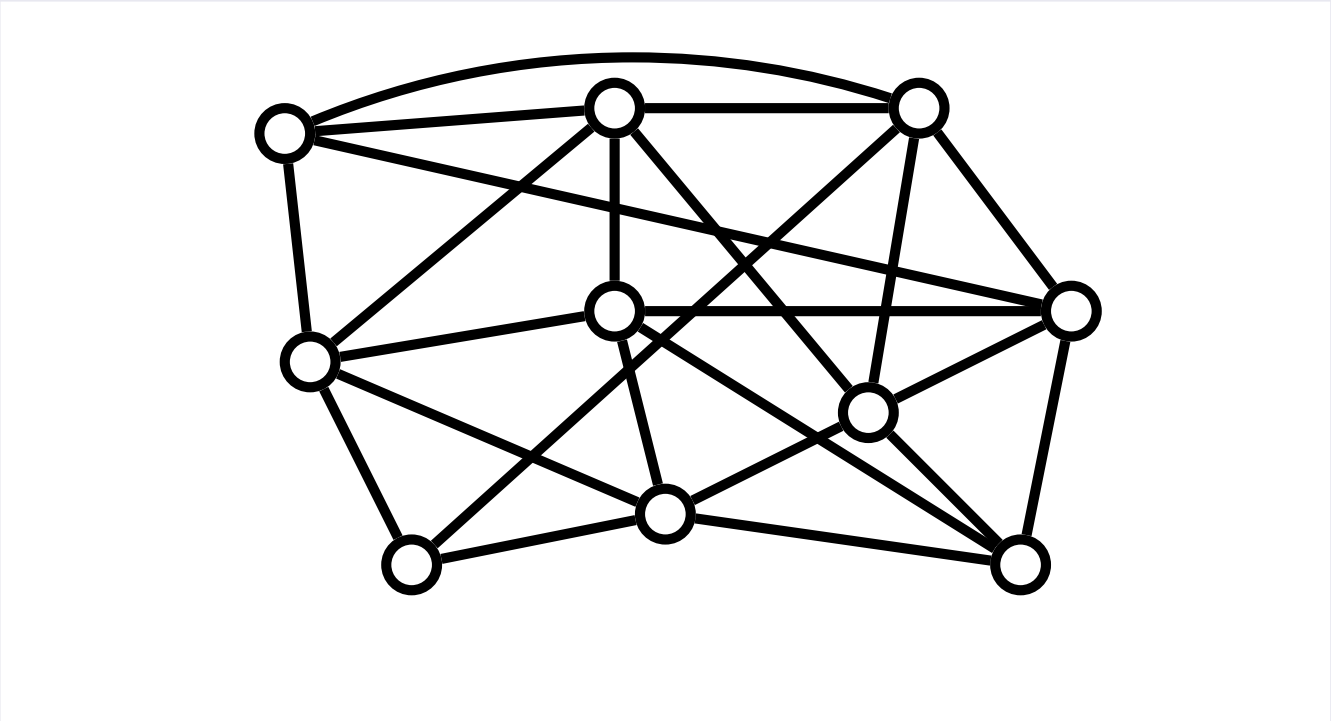
- We **cannot** get better than 7/8 for Max-3-SAT in sub-exp time (under ETH).
- We will therefore try to get something else:

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), (de la Vega '96)

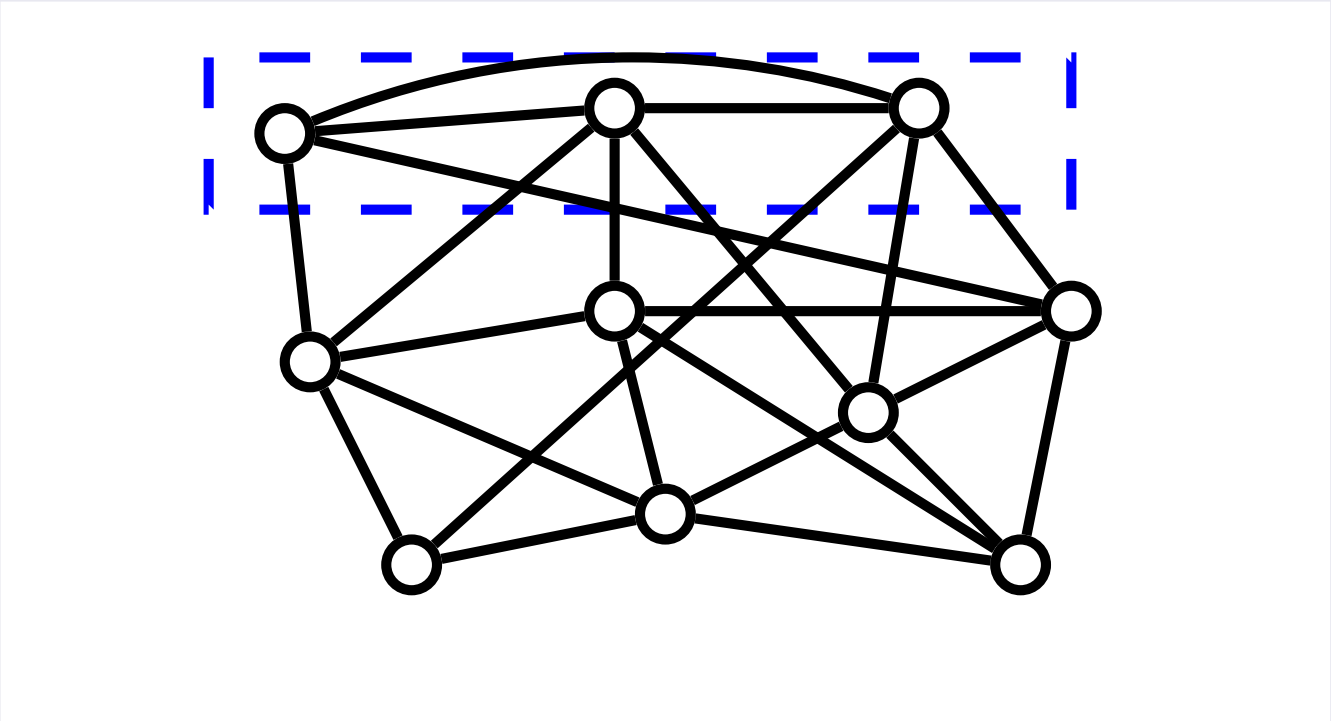
Strategy: We use our extra time to extend this island of tractability.

Basic scheme (Max Cut)



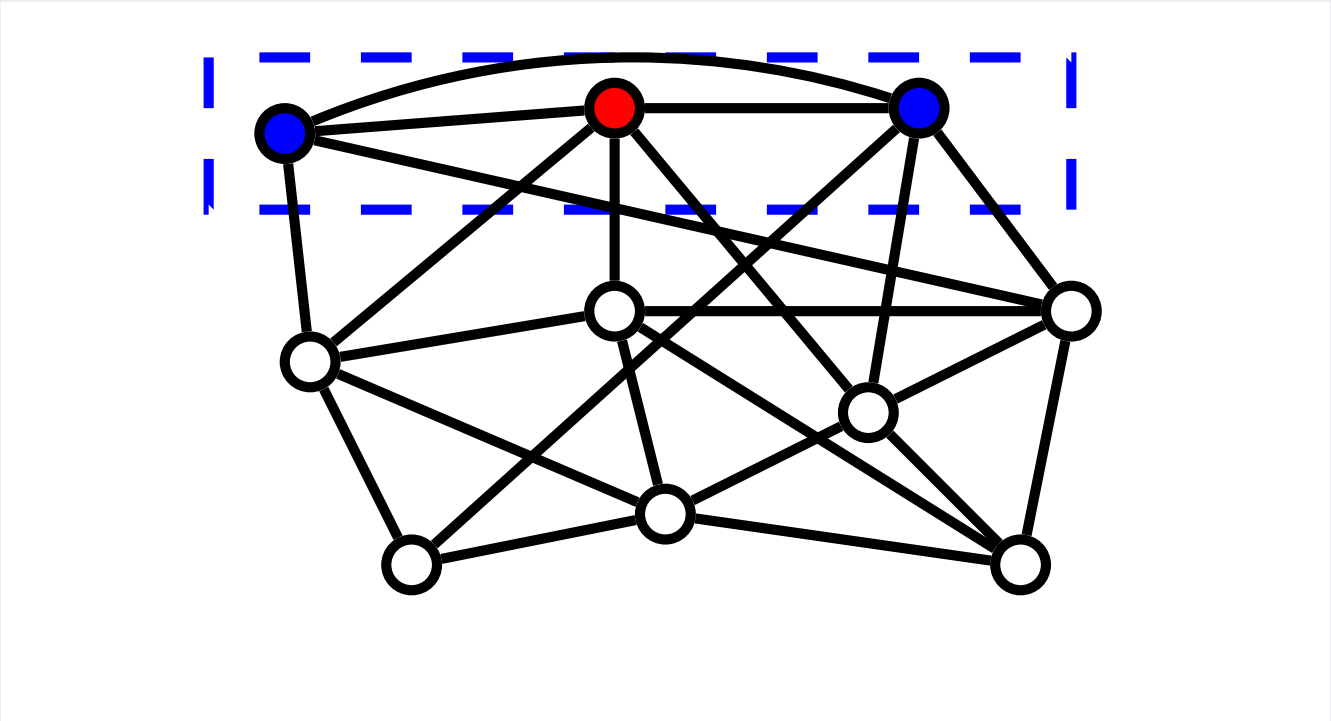
We are given a dense graph for which we want to find a large cut

Basic scheme (Max Cut)



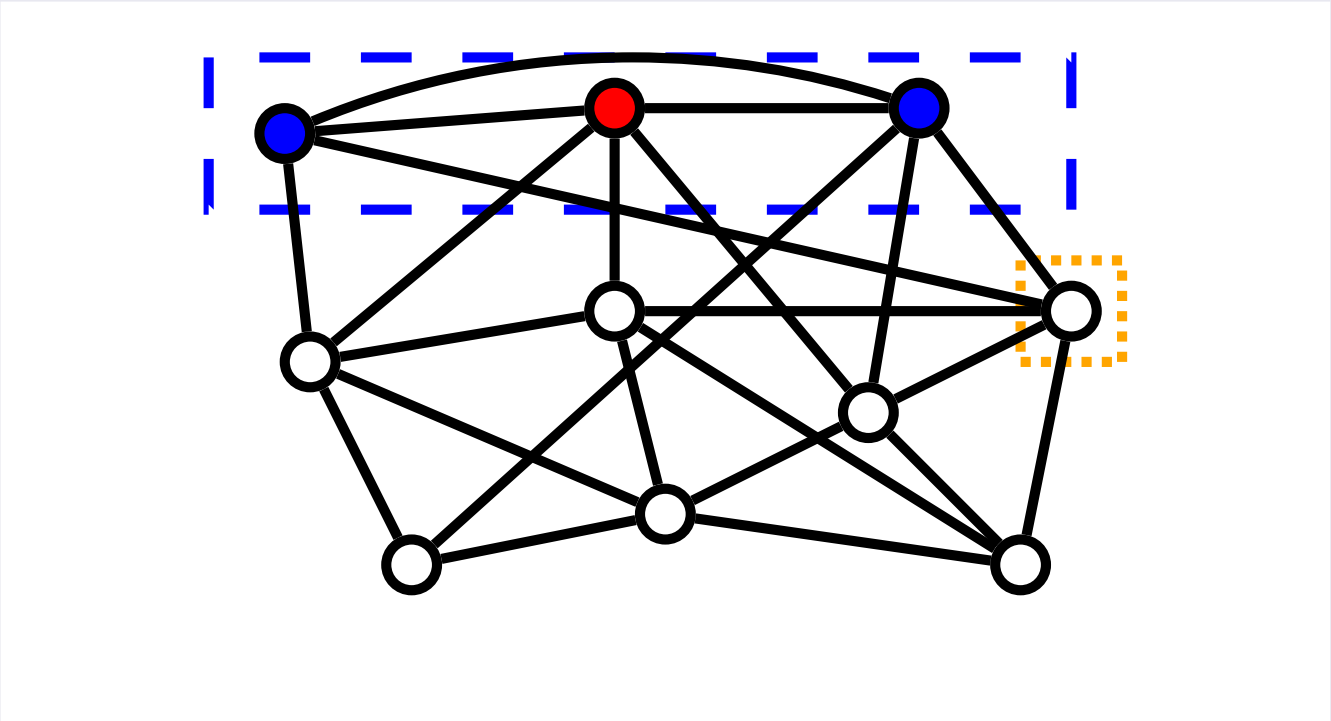
Randomly select a “sample” of its vertices

Basic scheme (Max Cut)



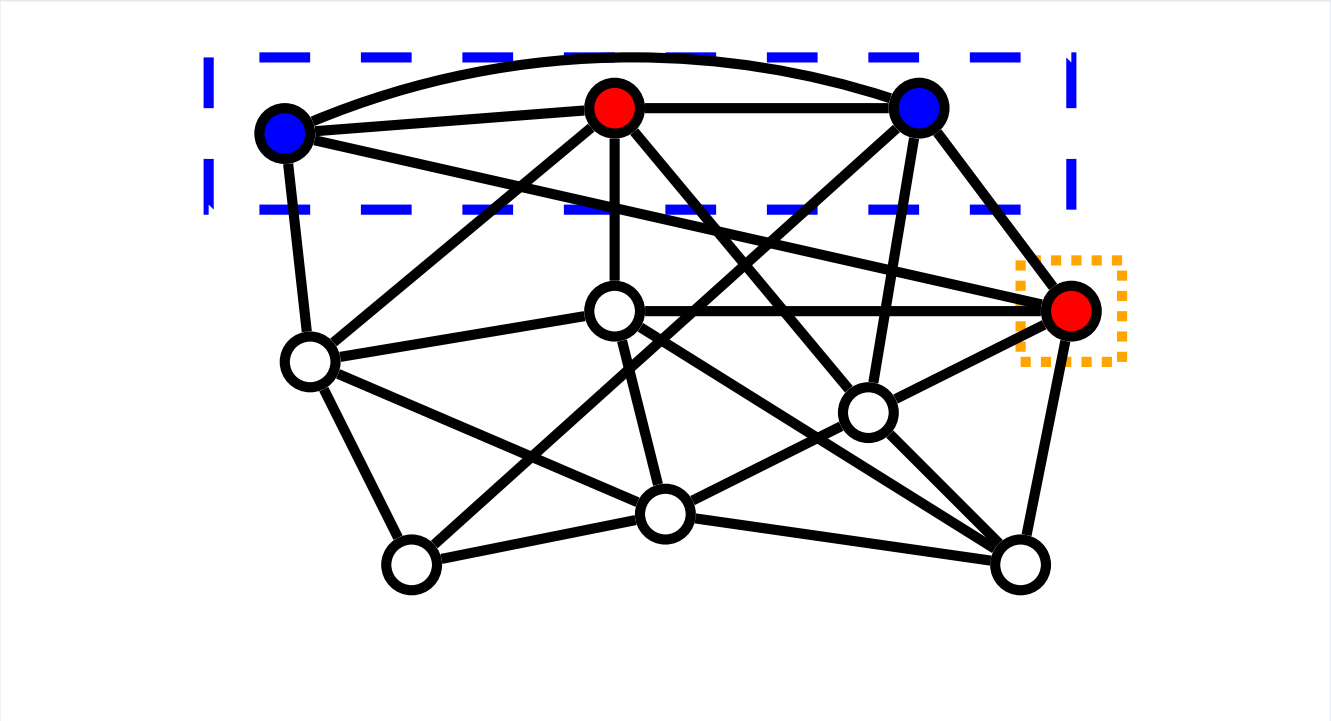
Guess their correct partition

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

(de la Vega '96)

Sub-exponential Extension (Max Cut)

Summary of algorithm:

- Estimate the neighborhood partitions using a sample
  - How can we do this if a graph is sparse?
- The rest works as previously...

Sub-exponential Extension (Max Cut)

Summary of algorithm:

- Estimate the neighborhood partitions using a sample
  - How can we do this if a graph is sparse?
- The rest works as previously...

Main idea: **Use larger sample**

Sub-exponential Extension (Max Cut)

Summary of algorithm:

- Estimate the neighborhood partitions using a sample
  - How can we do this if a graph is sparse?
- The rest works as previously...

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
- Running time: $\approx 2^{n^{1-\delta}}$

Sub-exponential Extension (Max Cut)

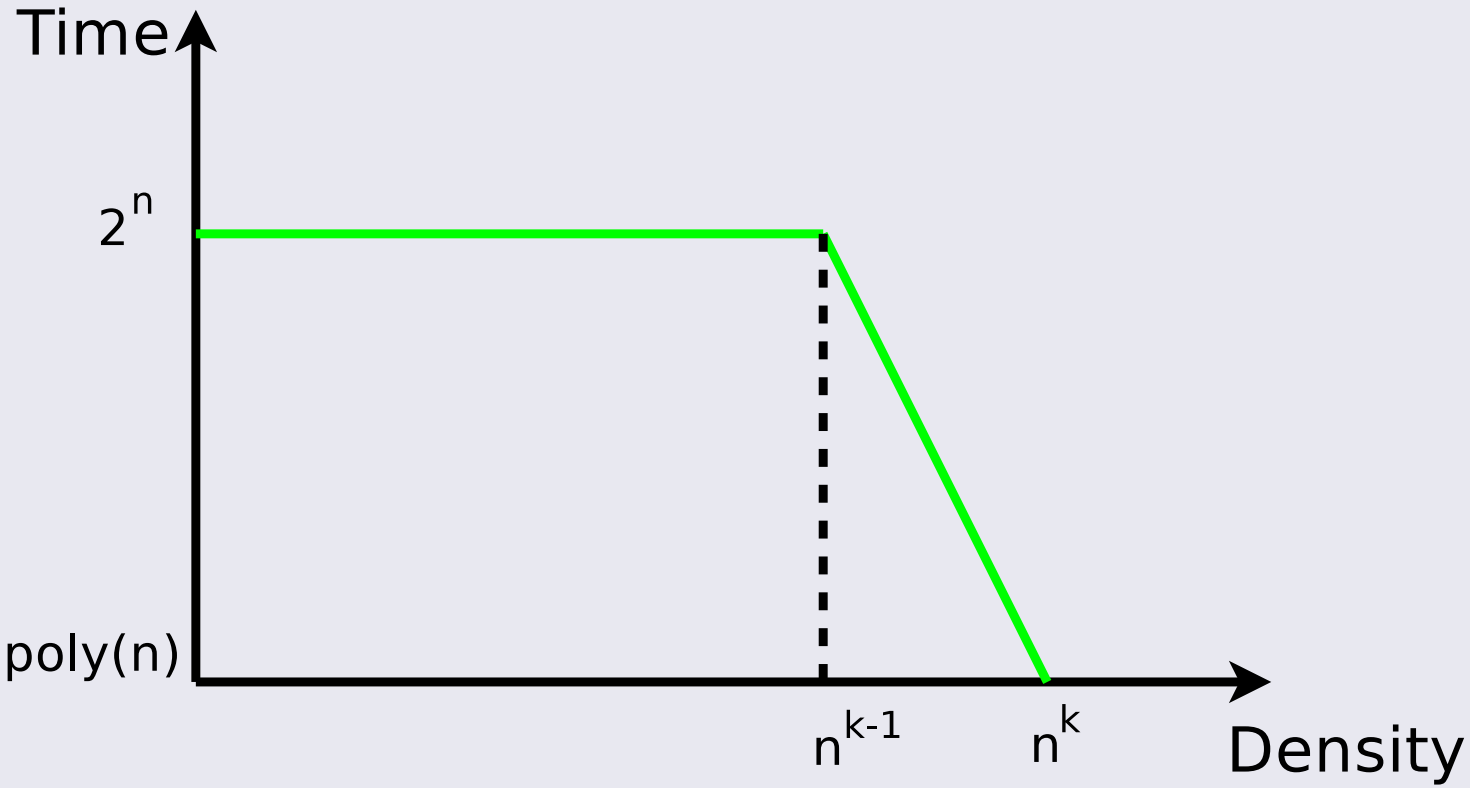
Summary of algorithm:

- Estimate the neighborhood partitions using a sample
  - How can we do this if a graph is sparse?
- The rest works as previously...

Main idea: **Use larger sample**

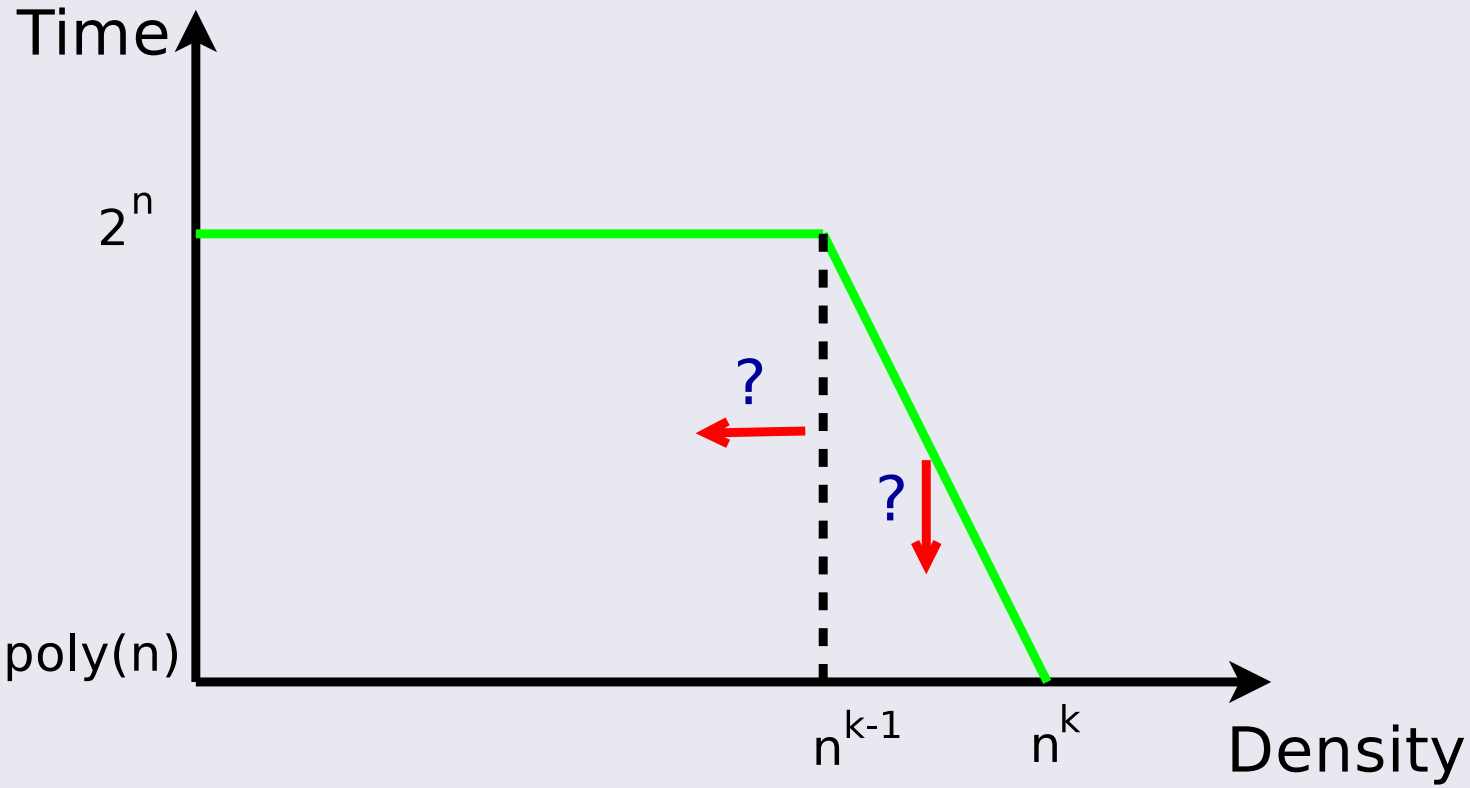
- Suppose graph has average degree $\Delta = n^\delta$
- We sample $\frac{n \log n}{\Delta} = n^{1-\delta} \log n$ vertices
- Running time: $\approx 2^{n^{1-\delta}}$
- Can be extended to Max-k-CSP!
- If $n^{k-1+\delta}$ clauses $\rightarrow 2^{n^{1-\delta}}$ time

Summary – with a picture



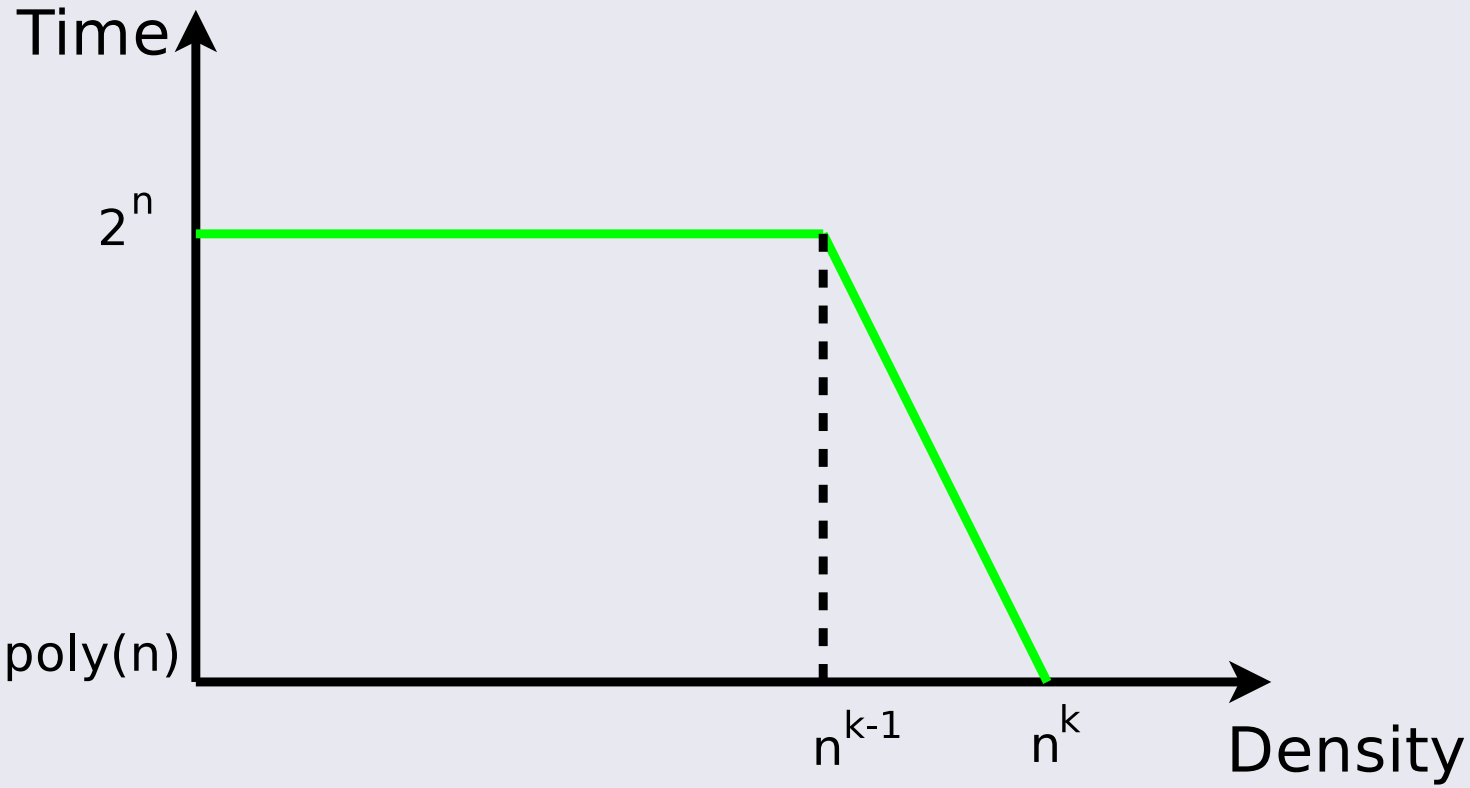
Complexity/Density trade-off for Max-k-CSP

Summary – with a picture



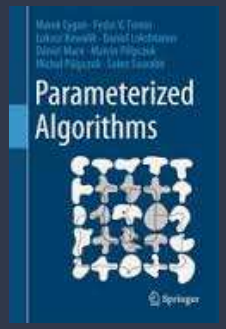
Possible Improvements? Faster? More general?

Summary – with a picture

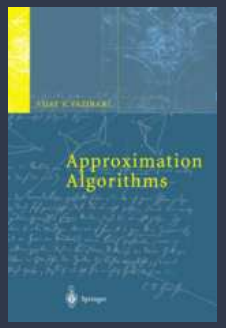


Both are tight! (under ETH)

Parameterized Approximation



+



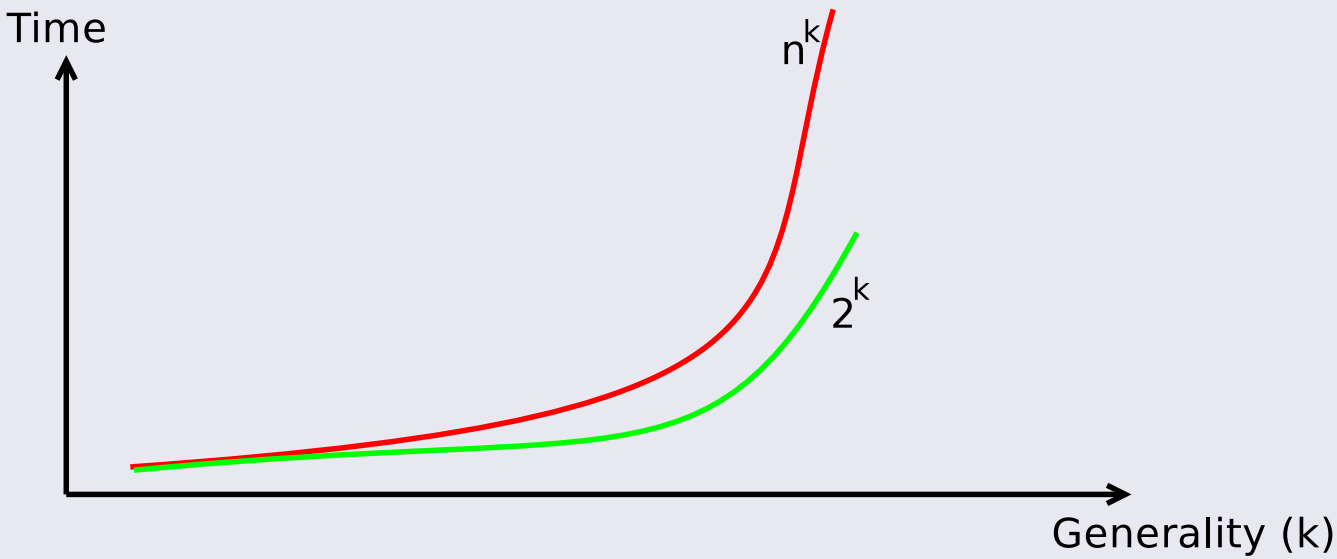
=

General Approach

- Parameterized Complexity == Trading Time for Generality
- General question: What is the best trade-off curve?
  - n^k vc. 2^k

General Approach

- Parameterized Complexity == Trading Time for Generality
- General question: What is the best trade-off curve?
  - n^k vc. 2^k



Does the trade-off curve become (much) better if we allow $(1+\epsilon)$ error?

Win/Win – Max-SAT

Joint work with:

Holger Dell

Eunjung Kim

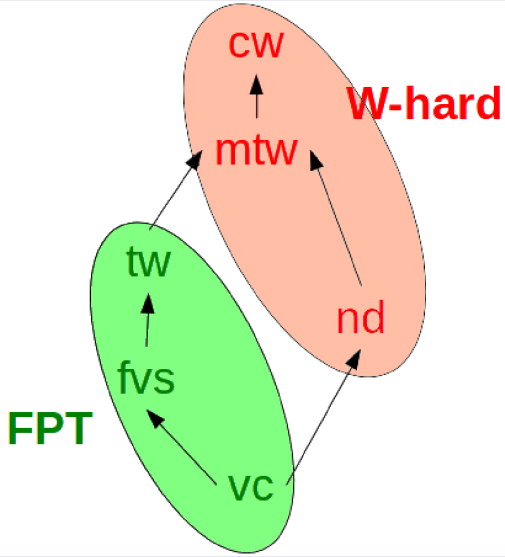
Valia Mitsou

Tobias Moemke

"Complexity and Approximability of Parameterized Max-CSPs", IPEC '15

Win/Win – Max-SAT

Problem: Max-SAT with structural graph parameters



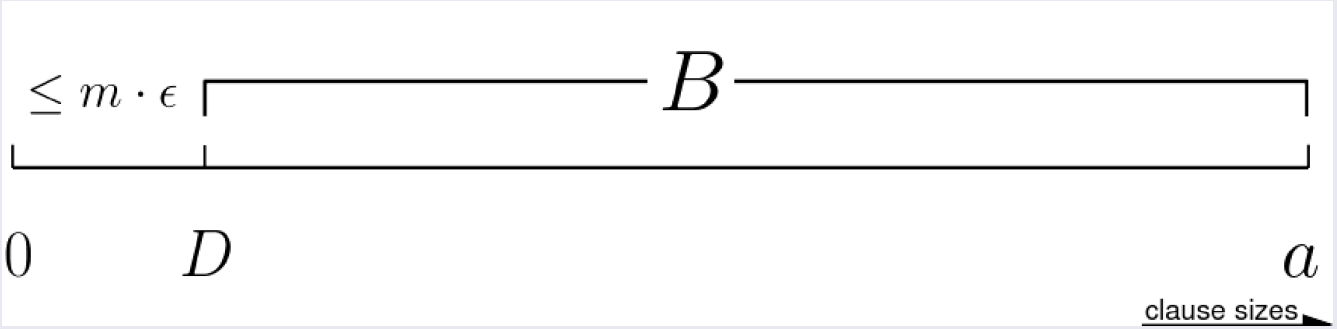
- Solvable in 2^k for treewidth
- Needs n^k for clique-width
- (APX-hard in general)
- **Goal:** Faster algorithm for more general case (clique-width), for $(1+\epsilon)$-approximation error.

Win/Win – Max-SAT

Algorithmic Strategy: **Win/Win**

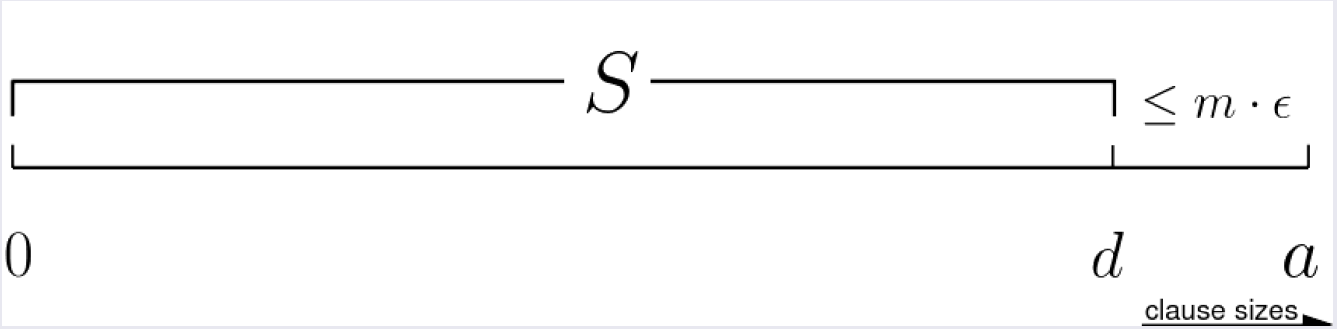
Win/Win – Max-SAT

Algorithmic Strategy: **Win/Win**



- If (almost) all clauses large $\rightarrow$ Easy!
  - Set everything randomly.

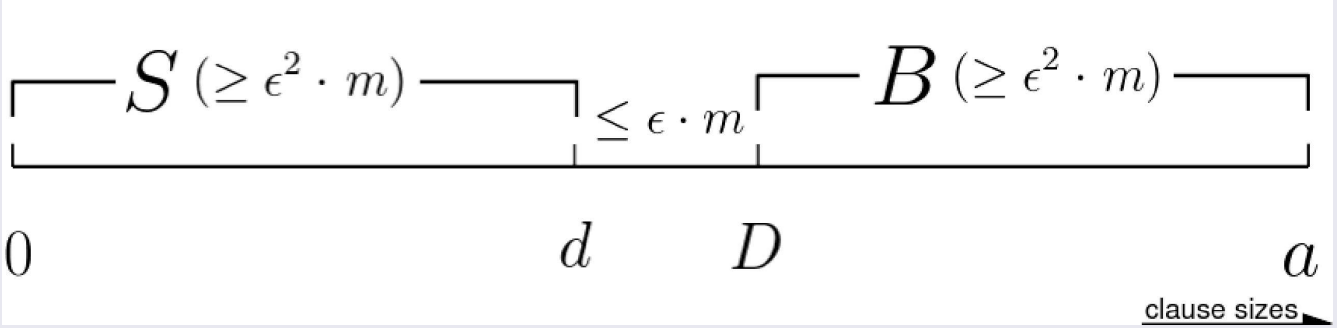
Algorithmic Strategy: **Win/Win**



- If (almost) all clauses large $\rightarrow$ Easy!
- If (almost) all clauses small $\rightarrow$ Easy!
  - Instance has small treewidth.

Win/Win – Max-SAT

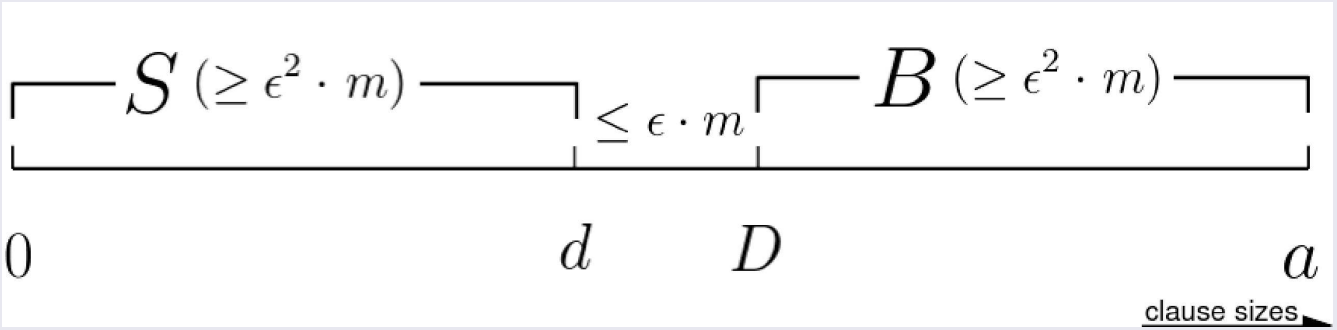
Algorithmic Strategy: **Win/Win**



- If (almost) all clauses large $\rightarrow$ Easy!
- If (almost) all clauses small $\rightarrow$ Easy!
- If many large and many small $\rightarrow$ Reduce to previous cases
  - There exists variable appearing overwhelmingly in large clauses.
  - Set it randomly, repeat.

Win/Win – Max-SAT

Algorithmic Strategy: **Win/Win**



- If (almost) all clauses large $\rightarrow$ Easy!
- If (almost) all clauses small $\rightarrow$ Easy!
- If many large and many small $\rightarrow$ Reduce to previous cases

- Bottleneck: FPT algorithm for treewidth (small clause case)
- $(1+\epsilon)$-approximation
- Running time: $2^{O(k)}$, where constant depends on ϵ
- **Contrast:** Exact algorithm needs n^k.

FPT Rounding – Ordered Weighted Assignment

Joint work with:

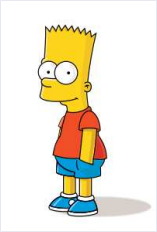
Julien Lesca

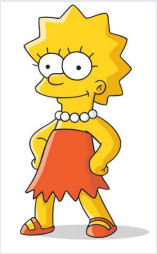
Florian Sikora

FPT Rounding – Ordered Weighted Assignment



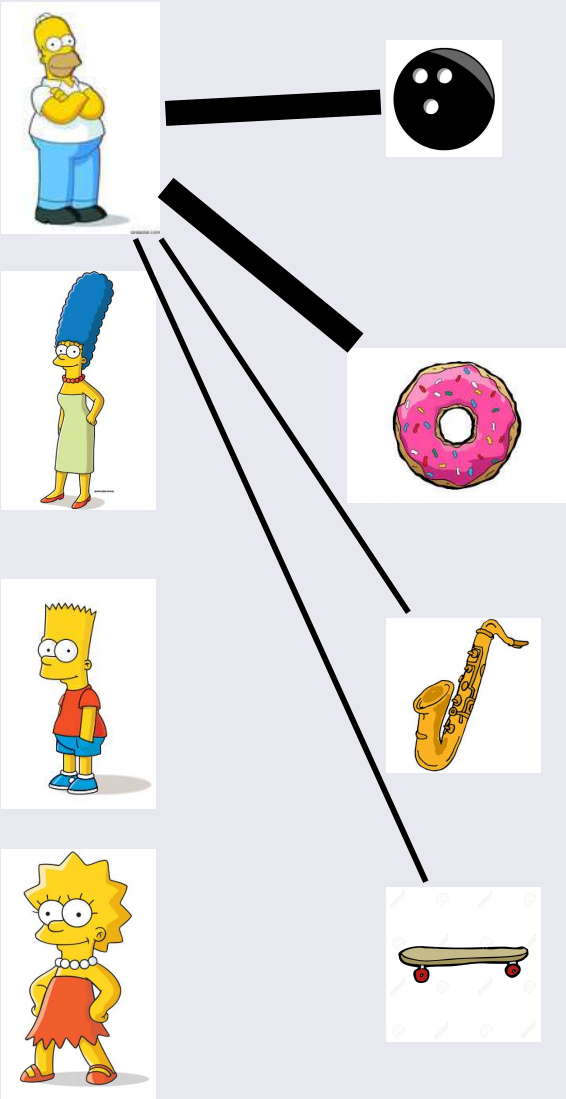






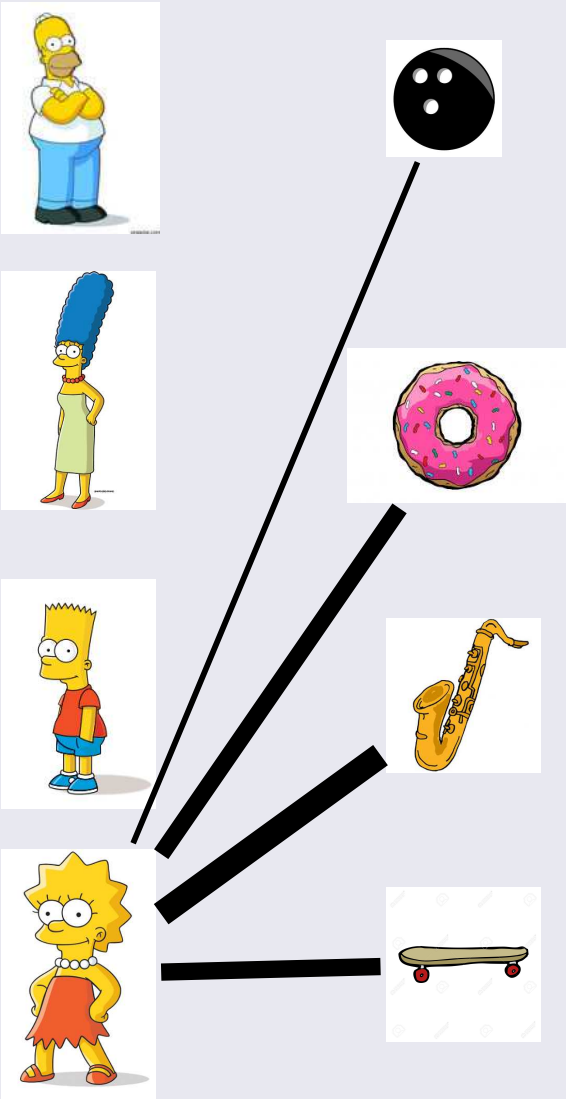

- n agents
- n items
- Each agent has different utility for each item
- Maximize **Social Welfare** (not necessarily sum)

FPT Rounding – Ordered Weighted Assignment



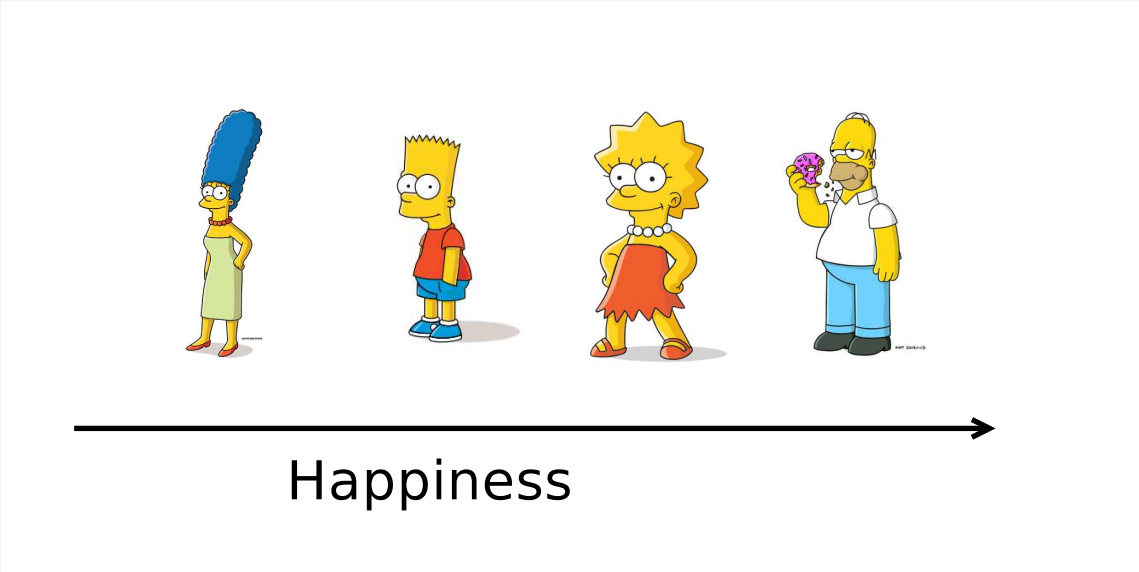
- n agents
- n items
- Each agent has different utility for each item
- Maximize **Social Welfare** (not necessarily sum)

FPT Rounding – Ordered Weighted Assignment



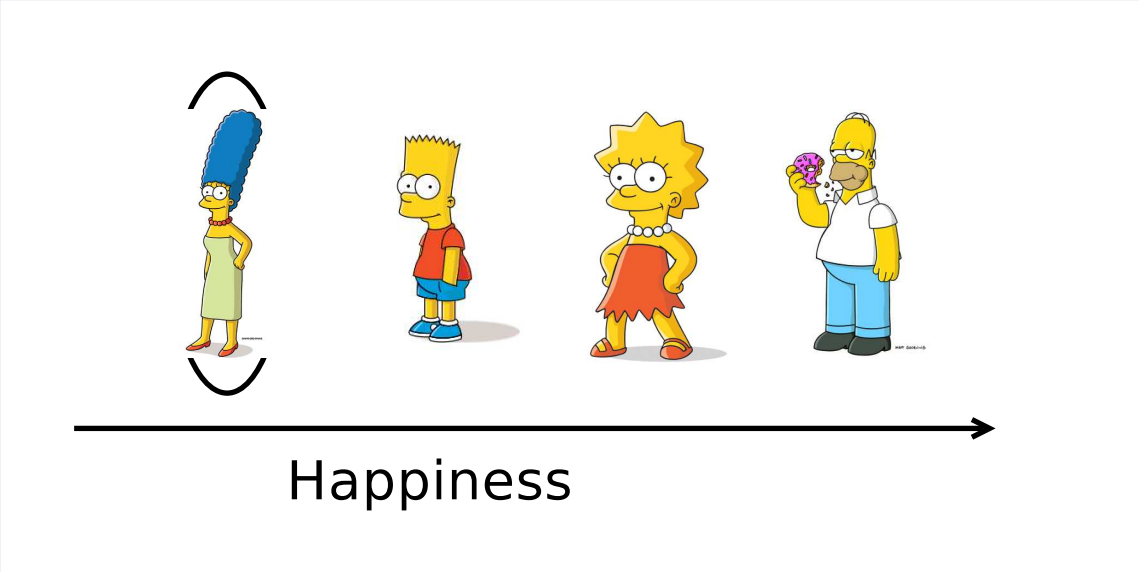
- n agents
- n items
- Each agent has different utility for each item
- Maximize **Social Welfare** (not necessarily sum)

FPT Rounding – Ordered Weighted Assignment



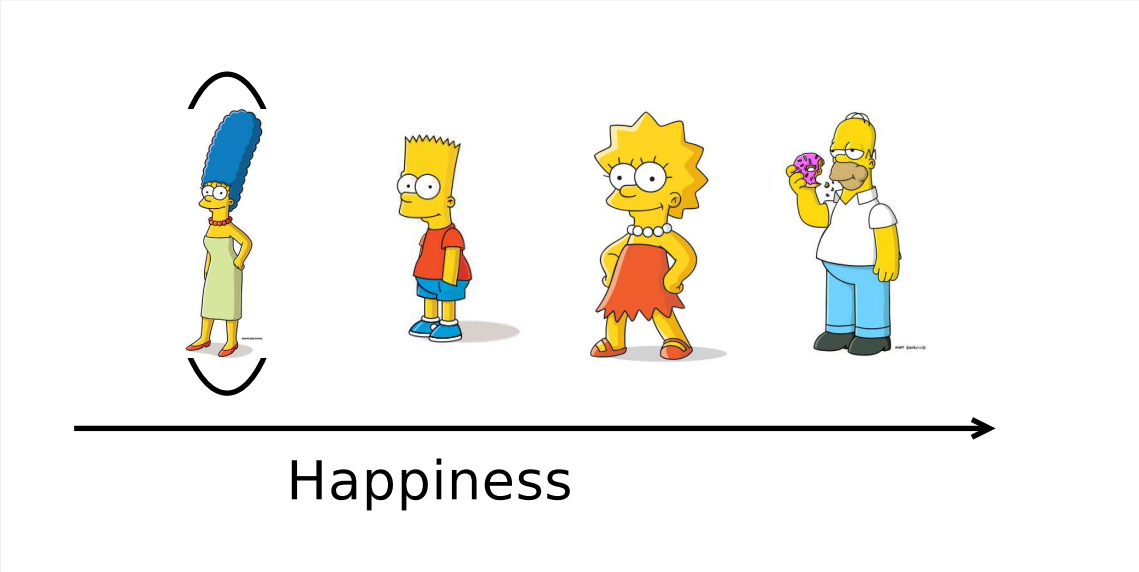
- Usual objective: Maximize sum of utilities
  - Maximum Matching ($\in$ P)

FPT Rounding – Ordered Weighted Assignment



- Usual objective: Maximize sum of utilities
  - Maximum Matching ($\in$ P)
- More equitable objective: Maximize min utility
  - Guess min utility, Perfect Matching ($\in$ P)

FPT Rounding – Ordered Weighted Assignment



- Usual objective: Maximize sum of utilities
  - Maximum Matching ($\in$ P)
- More equitable objective: Maximize min utility
  - Guess min utility, Perfect Matching ($\in$ P)

Something between the two extremes?

OWA continued

Objective: Maximize the total utility of **the bottom 50%** of agents

- Strikes a balance between two extremes

OWA continued

Objective: Maximize the total utility of **the bottom 50%** of agents

- Strikes a balance between two extremes
- Solvable in P
  - Guess max utility in bottom 50%
  - Reduce higher utilities
  - Maximum matching

OWA continued

Objective: Maximize the total utility according to **social priorities**

Position	0-20%	20-40%	40-60%	60-80%	80-100%
Weight	14	12	7	5	1

- Algorithm for k groups
  - Guess max utility in each group
  - Adjust utilities
  - Maximum matching
- Time complexity: n^{2k}
- Optimal! (no $n^{o(k)}$)

OWA Rounding

Basic idea: “Round down” large utilities to decrease search space

- Suppose all utilities $n^{O(1)}$ (wlog)
- Replace each value u_{ij} with

$$(1+\epsilon)^{\lfloor \log_{1+\epsilon} u_{ij} \rfloor}$$

OWA Rounding

Basic idea: “Round down” large utilities to decrease search space

- Suppose all utilities $n^{O(1)}$ (wlog)
- Replace each value u_{ij} with

$$(1+\epsilon)^{\lfloor \log_{1+\epsilon} u_{ij} \rfloor}$$

- No value distorted by more than a $(1+\epsilon)$ factor!
- Now only $\log_{1+\epsilon} n$ possible values to guess

OWA Rounding

Basic idea: “Round down” large utilities to decrease search space

- Suppose all utilities $n^{O(1)}$ (wlog)
- Replace each value u_{ij} with

$$(1+\epsilon)^{\lfloor \log_{1+\epsilon} u_{ij} \rfloor}$$

- No value distorted by more than a $(1+\epsilon)$ factor!
- Now only $\log_{1+\epsilon} n$ possible values to guess

- Running time: $O((\log n)^k \cdot \mathrm{poly}(n))$
- $(1+\epsilon)$-approximation!

- (Reminder:) Exact algorithm needs n^{2k}

Power Vertex Cover – Rounding

Joint work with:

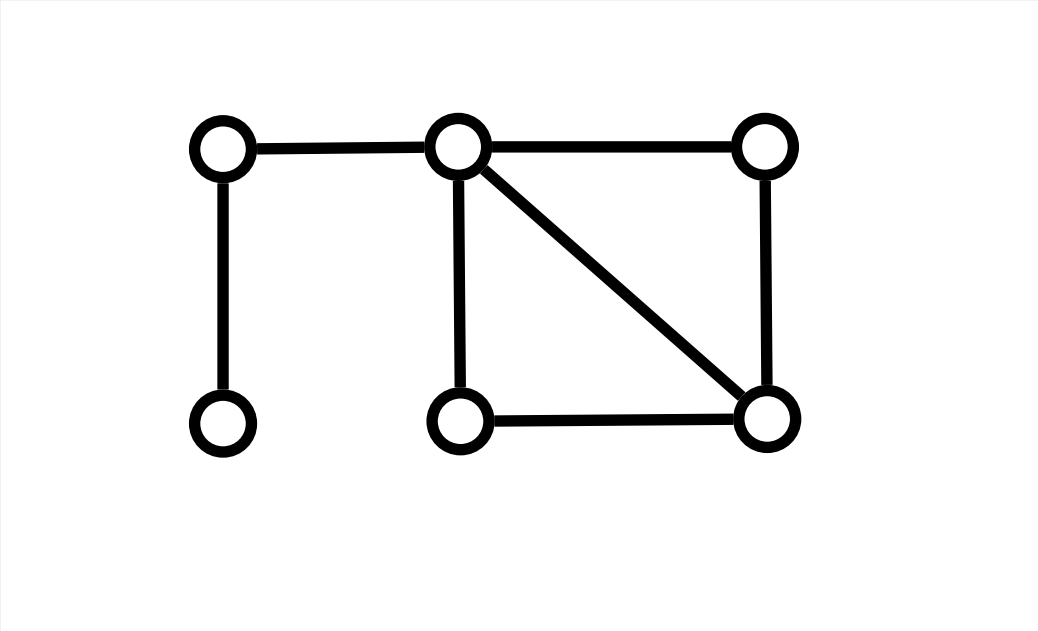
Eric Angel

Evripidis Bampis

Bruno Escoffier

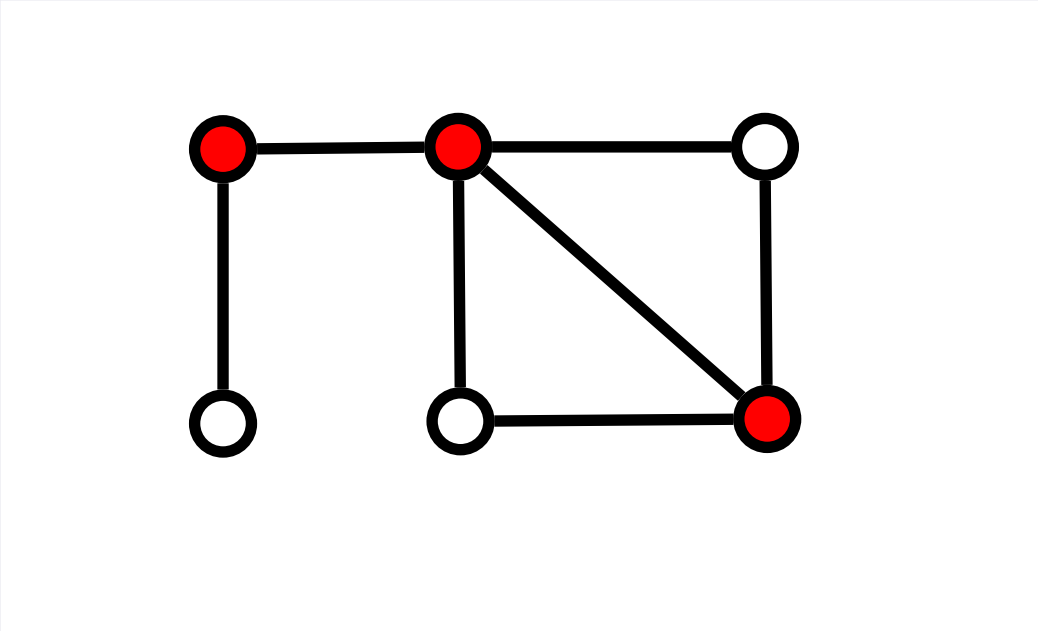
"Parameterized Power Vertex Cover", WG '16

Power Vertex Cover – Rounding



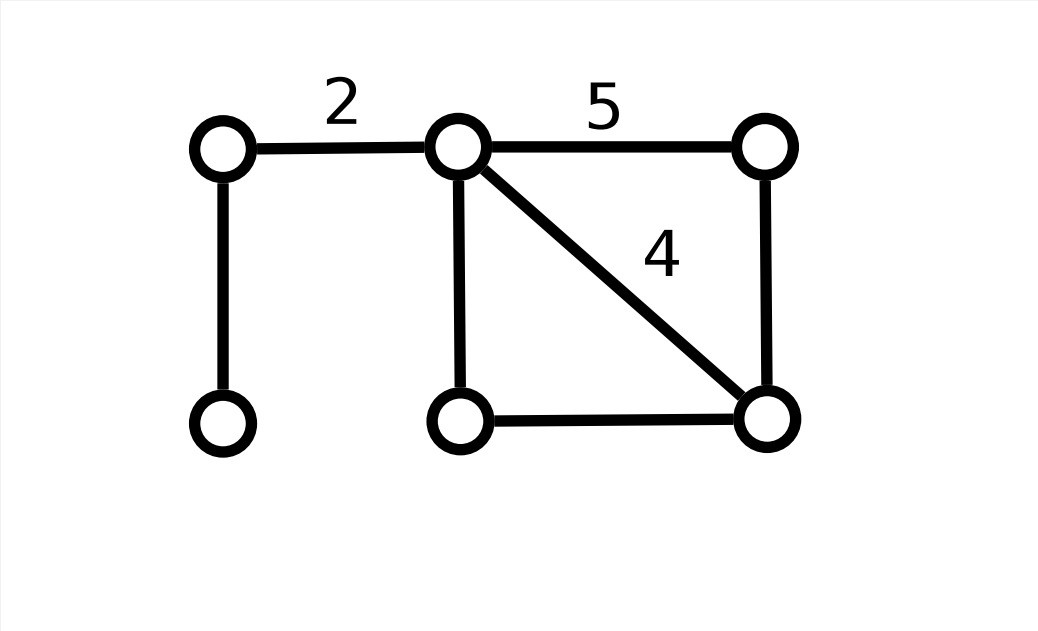
Vertex Cover: Select vertices that touch all edges

Power Vertex Cover – Rounding



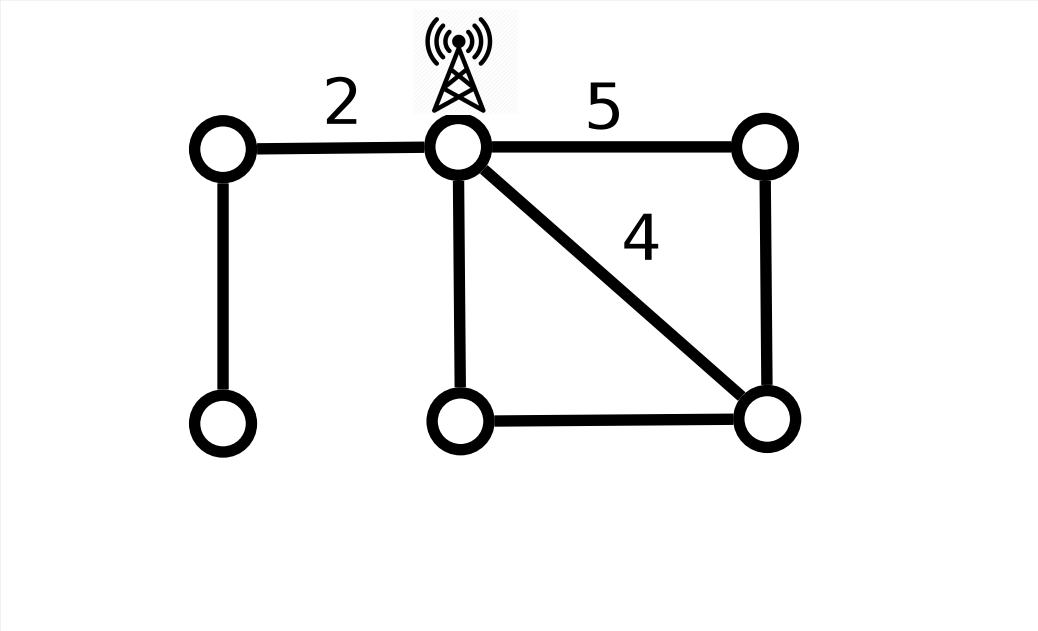
Vertex Cover: Select vertices that touch all edges

Power Vertex Cover – Rounding



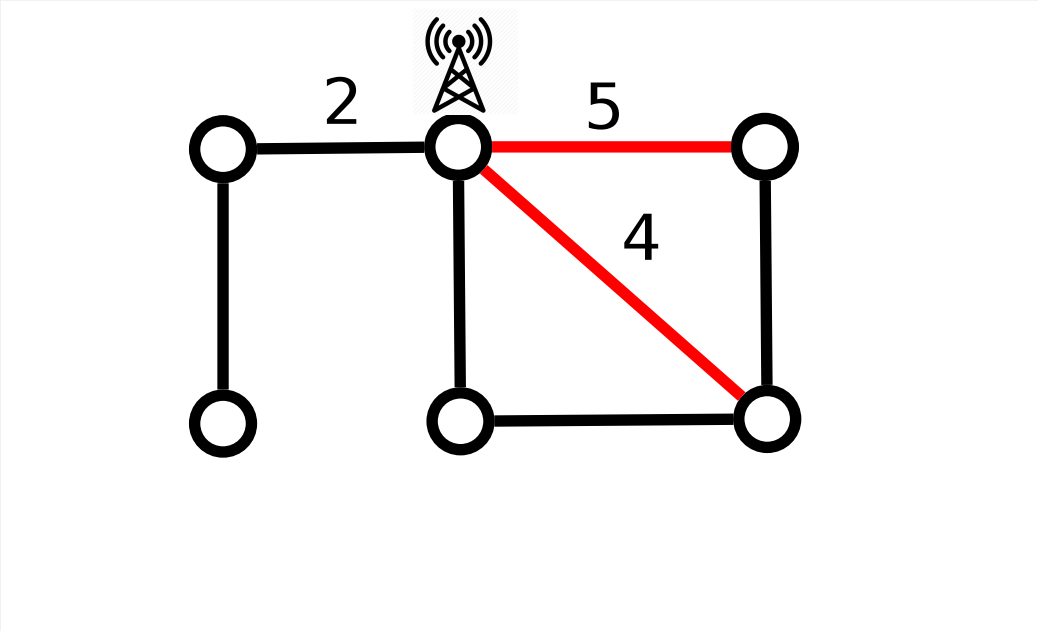
Power: Some edges **demand more power** to be covered

Power Vertex Cover – Rounding



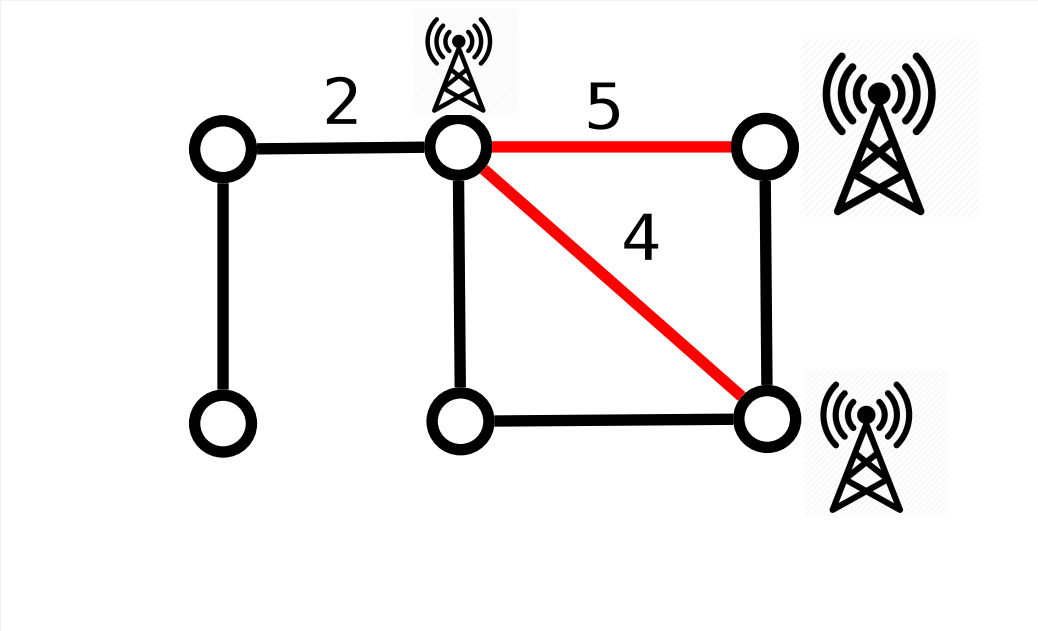
Power: Some edges **demand more power** to be covered

Power Vertex Cover – Rounding



Power: Some edges **demand more power** to be covered

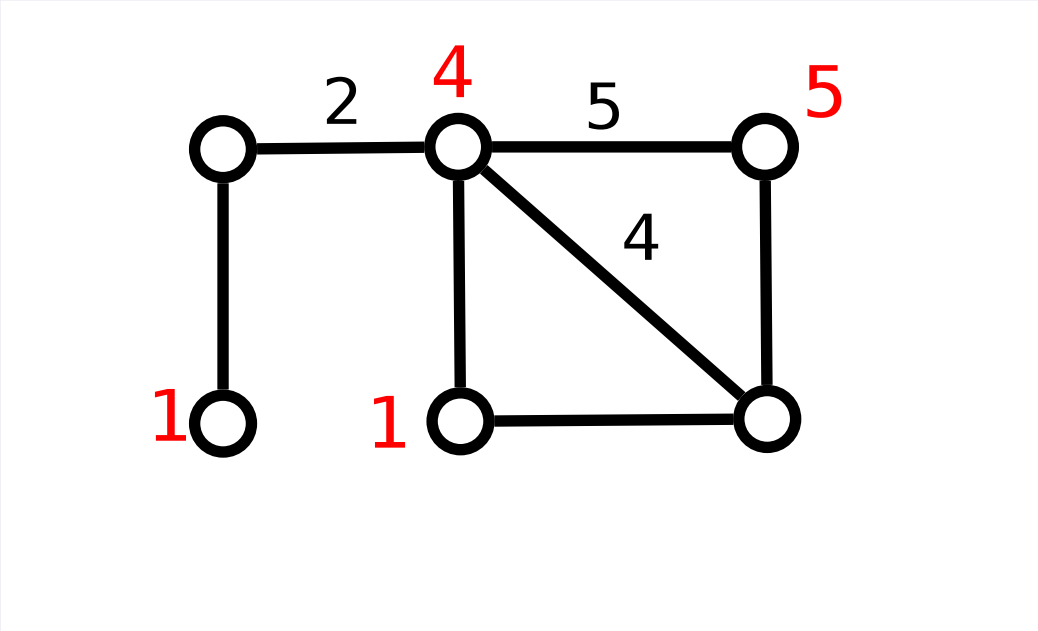
Power Vertex Cover – Rounding



Power Vertex Cover: Must decide which vertices get power

. . . and how much

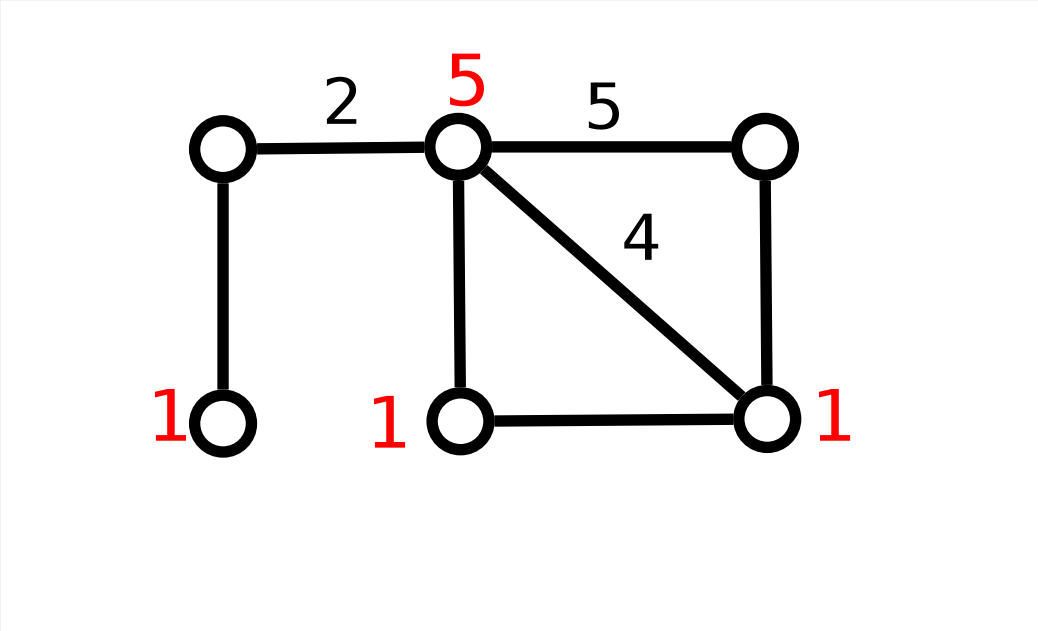
Power Vertex Cover – Rounding



Power Vertex Cover: Must decide which vertices get power

. . . **and how much**

Power Vertex Cover – Rounding



Formal Definition:

$$\min \sum p(v)$$

$$\max\{p(u), p(v)\} \geq d((u,v)) \;\; \forall (u,v) \in E$$

Motivation

- Applications to communication networks

Motivation

- Applications to communication networks ??

Motivation

- Applications to communication networks ??
- Interesting Generalization of Vertex Cover
  - Note: added **non-linear** constraint
    $\max\{p(u), p(v)\} \geq d((u, v)) \;\; \forall (u, v) \in E$
    - Compare: $p(u) + p(v) \geq d((u, v))$
  - Is this problem really different/harder from Vertex Cover?
    - Admits 2 approximation
    - In P for bipartite graphs [Angel et al. ISAAC '15]

Motivation

- Applications to communication networks ??
- Interesting Generalization of Vertex Cover
  - Note: added **non-linear** constraint
    $\max\{p(u), p(v)\} \geq d((u,v)) \;\; \forall (u,v) \in E$
    - Compare: $p(u) + p(v) \geq d((u,v))$
  - Is this problem really different/harder from Vertex Cover?
    - Admits 2 approximation
    - In P for bipartite graphs [Angel et al. ISAAC '15]
  - What about Parameterized algorithms?
    - Vertex Cover is **flagship** problem
    - Compare: Weighted VC, Capacitated VC, Connected VC, ...

Power Vertex Cover – Hardness

- We show hardness for treewidth (n^t lower bound)
- Consider the class $\mathrm{Trivial} + t$ vertices, result still holds.

Power Vertex Cover – Hardness

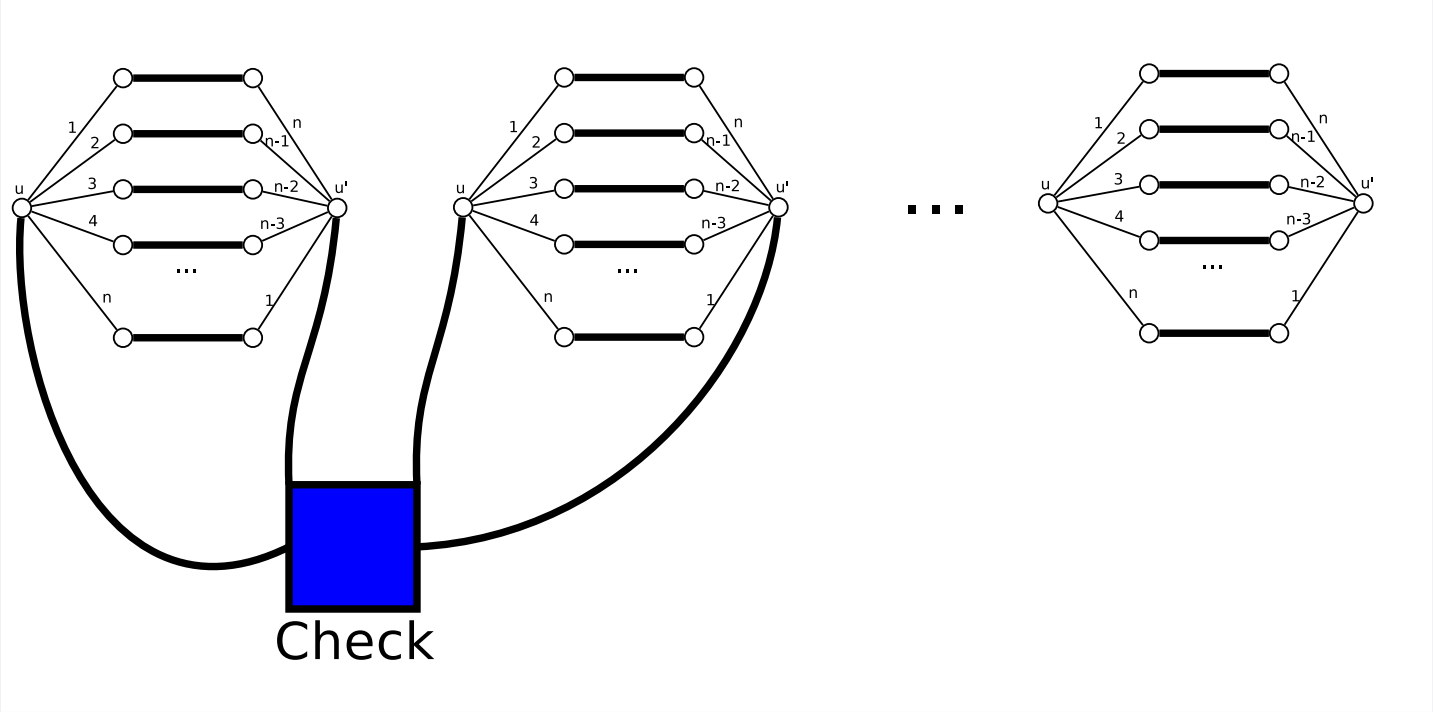
Theorem: There is no $n^{o(t)}$ algorithm for PVC (under ETH)

Power Vertex Cover – Hardness

Theorem: There is no $n^{o(t)}$ algorithm for PVC (under ETH)
Proof: Reduction from Multi-Colored Clique

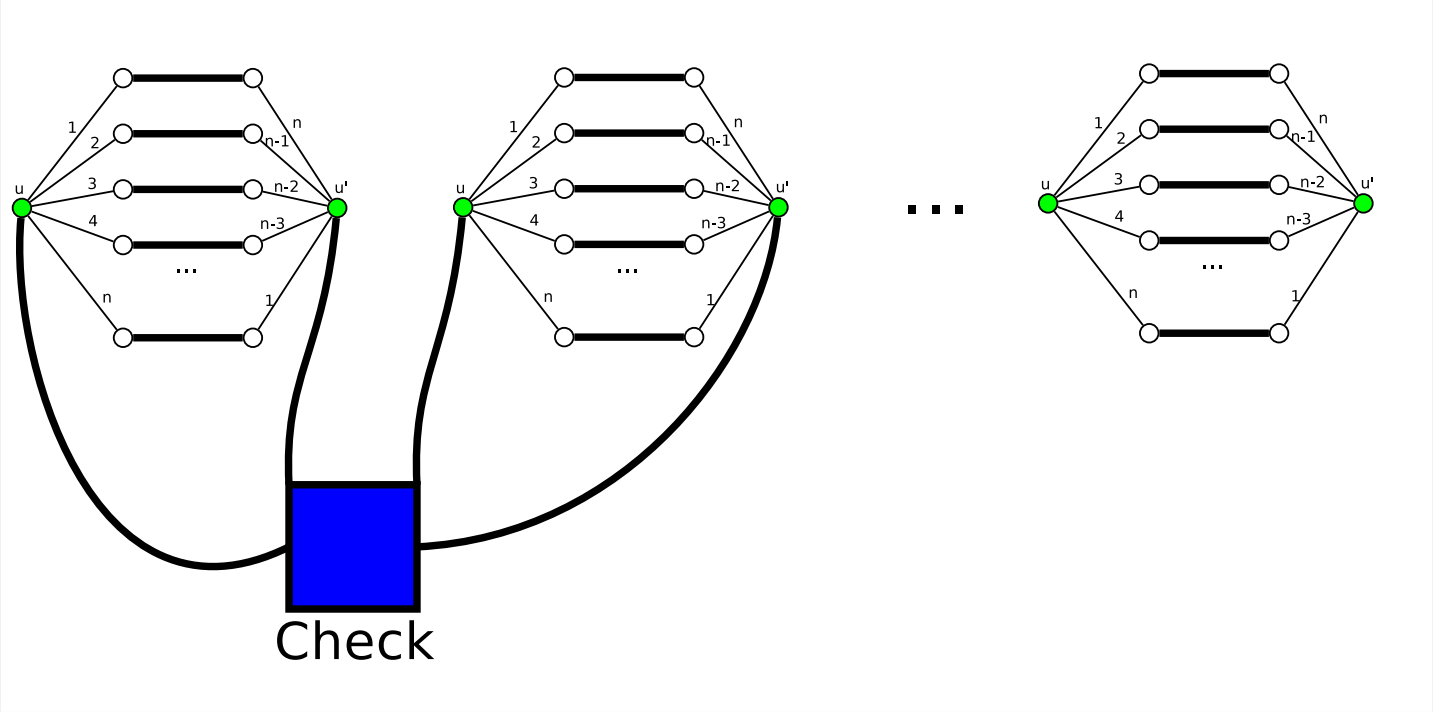
Power Vertex Cover – Hardness

Theorem: There is no $n^{o(t)}$ algorithm for PVC (under ETH)
Proof: Reduction from Multi-Colored Clique



Power Vertex Cover – Hardness

Theorem: There is no $n^{o(t)}$ algorithm for PVC (under ETH)
Proof: Reduction from Multi-Colored Clique



The only thing that matters is the power level of just k vertices!

Treewidth doesn't work!

Treewidth Algorithms

Treewidth doesn't work!

Actually it's not so bad...

Treewidth Algorithms

Easy **Exact** Algorithms

- $(\Delta + 1)^{tw} n$ time
- $(M + 1)^{tw} n$ time (M=maximum weight)

Main observation: Each vertex has limited number of reasonable power values.

(These running times are optimal)

Treewidth Algorithms

Easy **Exact** Algorithms

- $(\Delta + 1)^{tw} n$ time
- $(M + 1)^{tw} n$ time (M=maximum weight)

Main observation: Each vertex has limited number of reasonable power values.

(These running times are optimal)
Can we do better?

FPT **Approximation** Scheme

- $(M+1)^{tw}n$ time to solve exactly

FPT **Approximation** Scheme

- $(M+1)^{tw}n$ time to solve exactly
- Main idea: **Rounding**
  - Instead of power value p for each vertex store $\lfloor \log_{1+\epsilon}(p) \rfloor$
  - At most $\log M / \log(1+\epsilon)$ possible values
  - At most a $(1+\epsilon)$ factor from correct value
  - If $M = n^{O(1)}$ running time $(\log n/\epsilon)^{tw}$
  - (If not, easy: think Knapsack)

Treewidth Algorithms

FPT **Approximation** Scheme

- $(M+1)^{tw}n$ time to solve exactly
- Main idea: **Rounding**
  - Instead of power value p for each vertex store $\lfloor \log_{1+\epsilon}(p) \rfloor$
  - At most $\log M / \log(1+\epsilon)$ possible values
  - At most a $(1+\epsilon)$ factor from correct value
  - If $M = n^{O(1)}$ running time $(\log n/\epsilon)^{tw}$
  - (If not, easy: think Knapsack)

Bottom line: Fast FPT algorithm for W-hard problem, only $(1+\epsilon)$ error!
(This is part of a more general technique [L. ICALP '14])

Huge but Soft Lower Bounds

Temporal Spanning Trees

Joint work with:

Dimitris Fotakis

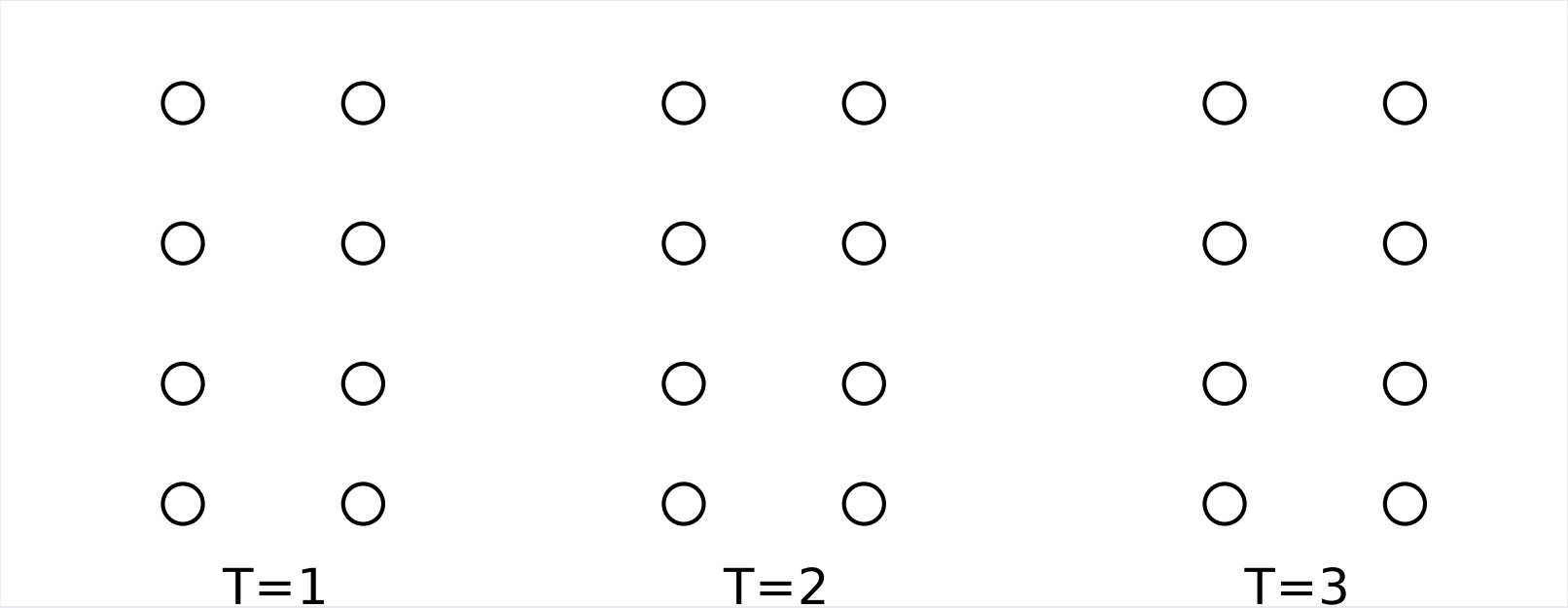
Vangelis Paschos

Temporal Spanning Trees

Input: T copies of the same graph, with different edge weights
Output: T spanning trees
Objective: Min weight cost + buying cost

Temporal Spanning Trees

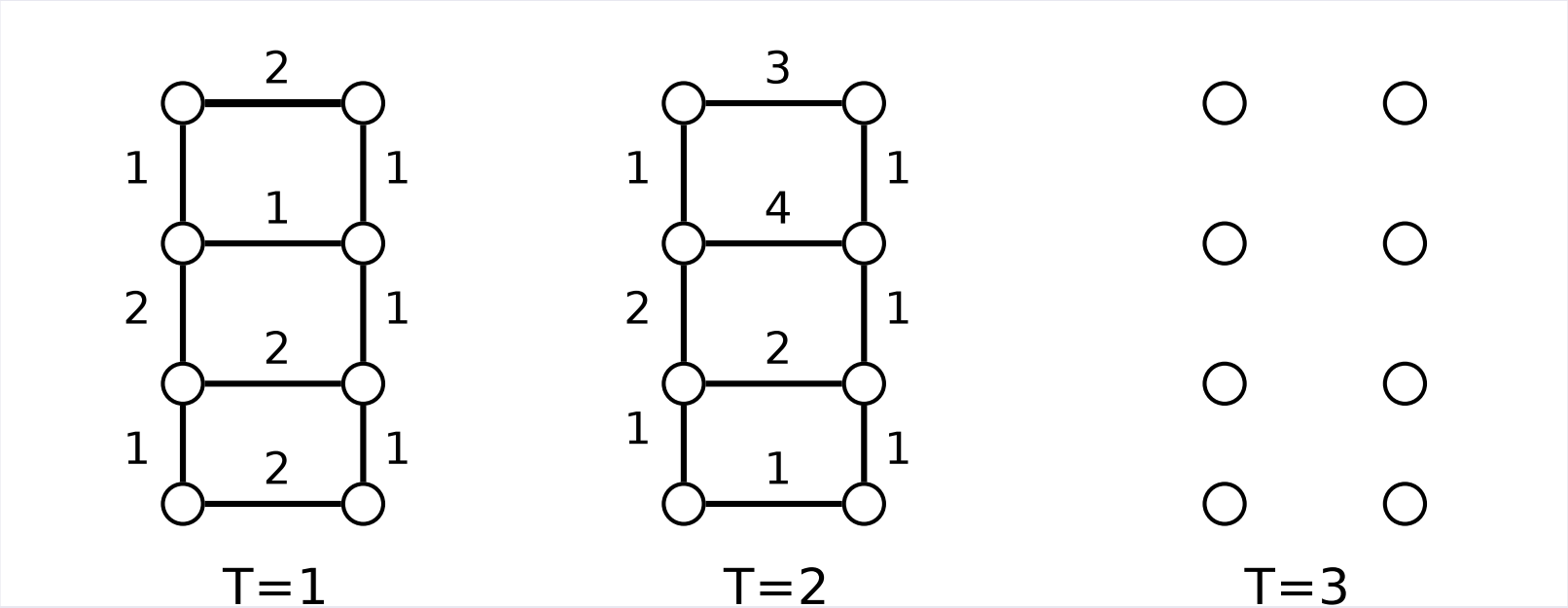
Input: T copies of the same graph, with different edge weights
Output: T spanning trees
Objective: Min weight cost + buying cost



Example: $T = 3$, buy$= 1$

Temporal Spanning Trees

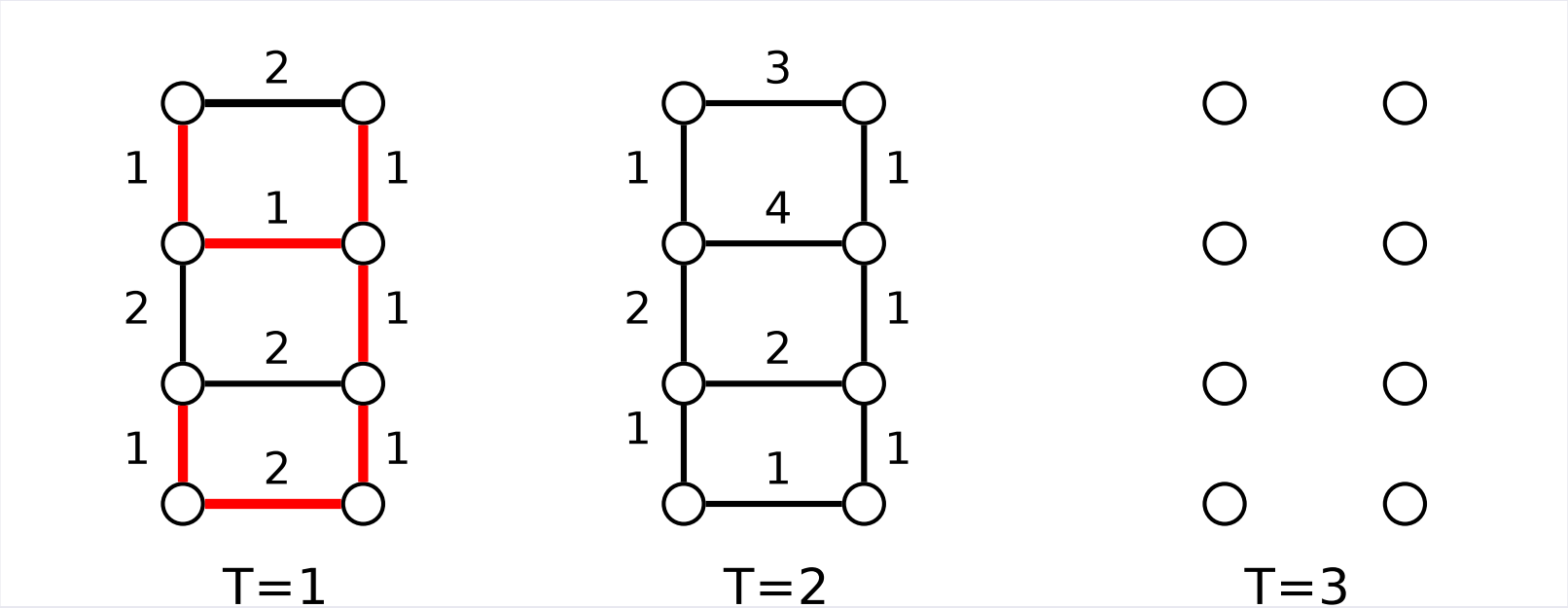
Input: T copies of the same graph, with different edge weights
Output: T spanning trees
Objective: Min weight cost + buying cost



Example: $T = 3$, buy$= 1$

Temporal Spanning Trees

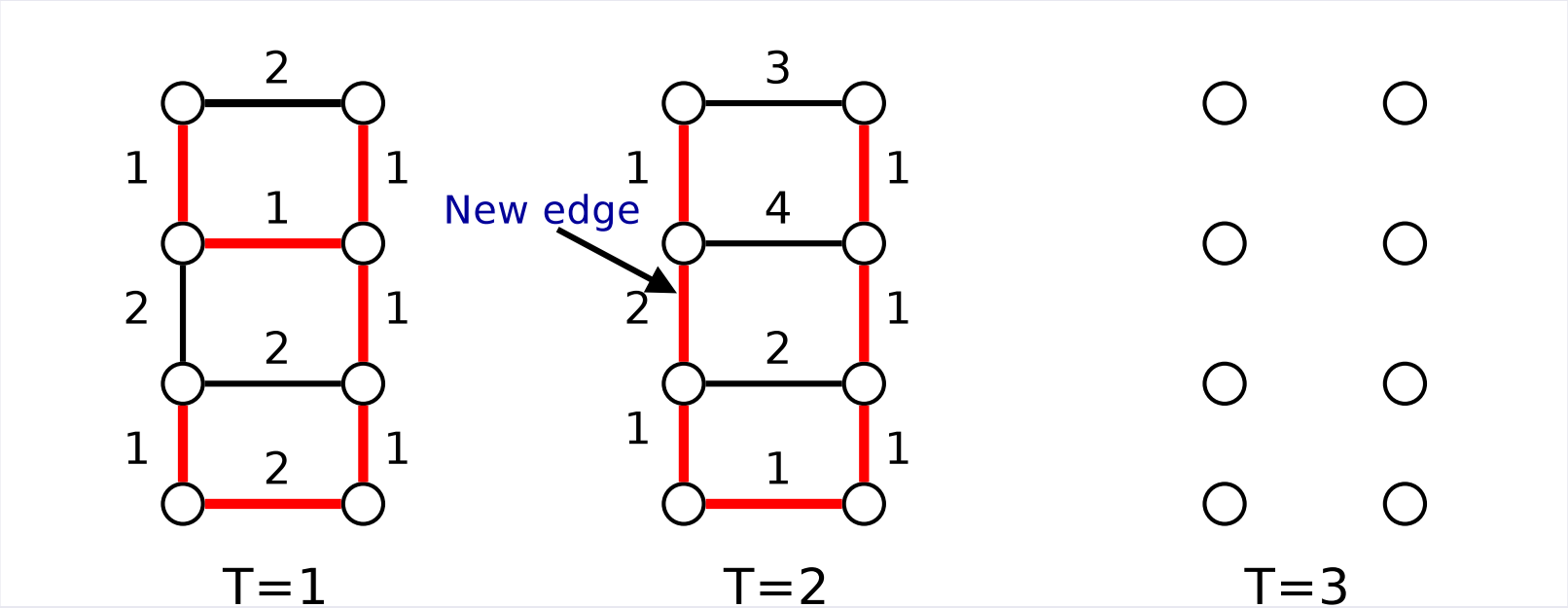
Input: T copies of the same graph, with different edge weights
Output: T spanning trees
Objective: Min weight cost + buying cost



Example: $T = 3$, buy$= 1$
Cost = 8

Temporal Spanning Trees

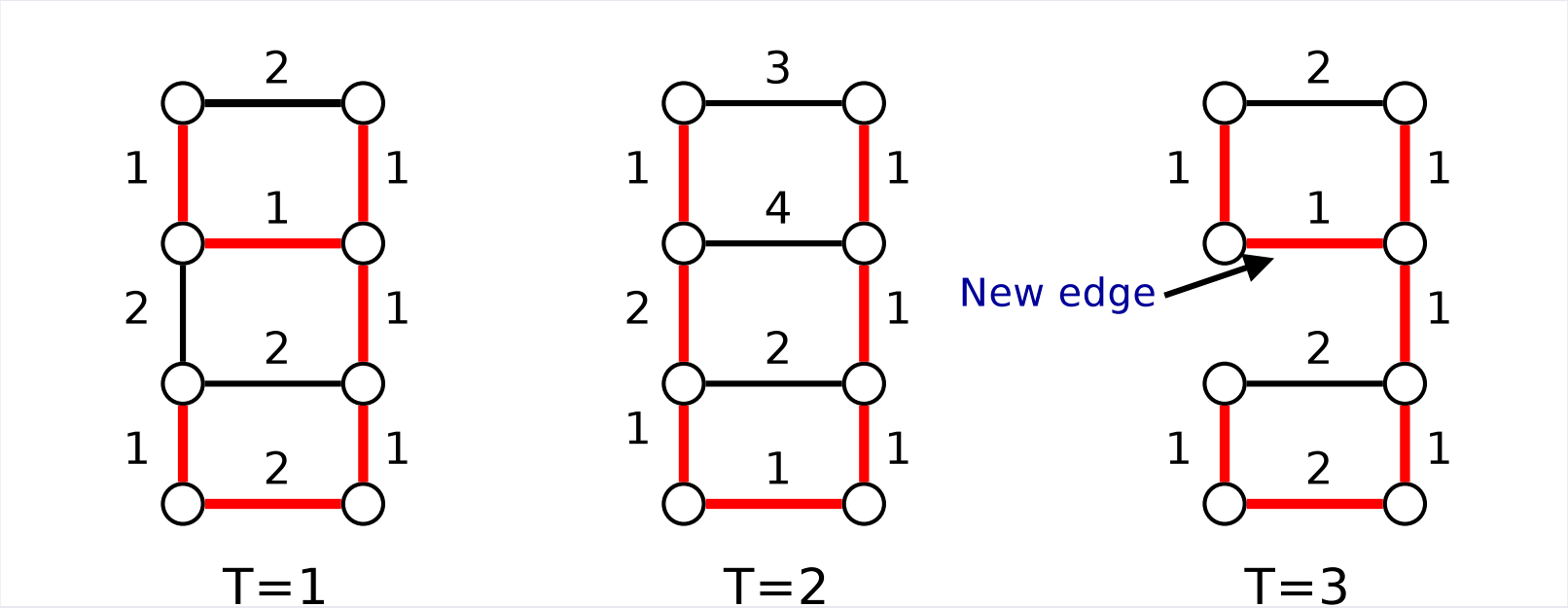
Input: T copies of the same graph, with different edge weights
Output: T spanning trees
Objective: Min weight cost + buying cost



Example: $T = 3$, buy$= 1$
Cost = 8+8+1

Temporal Spanning Trees

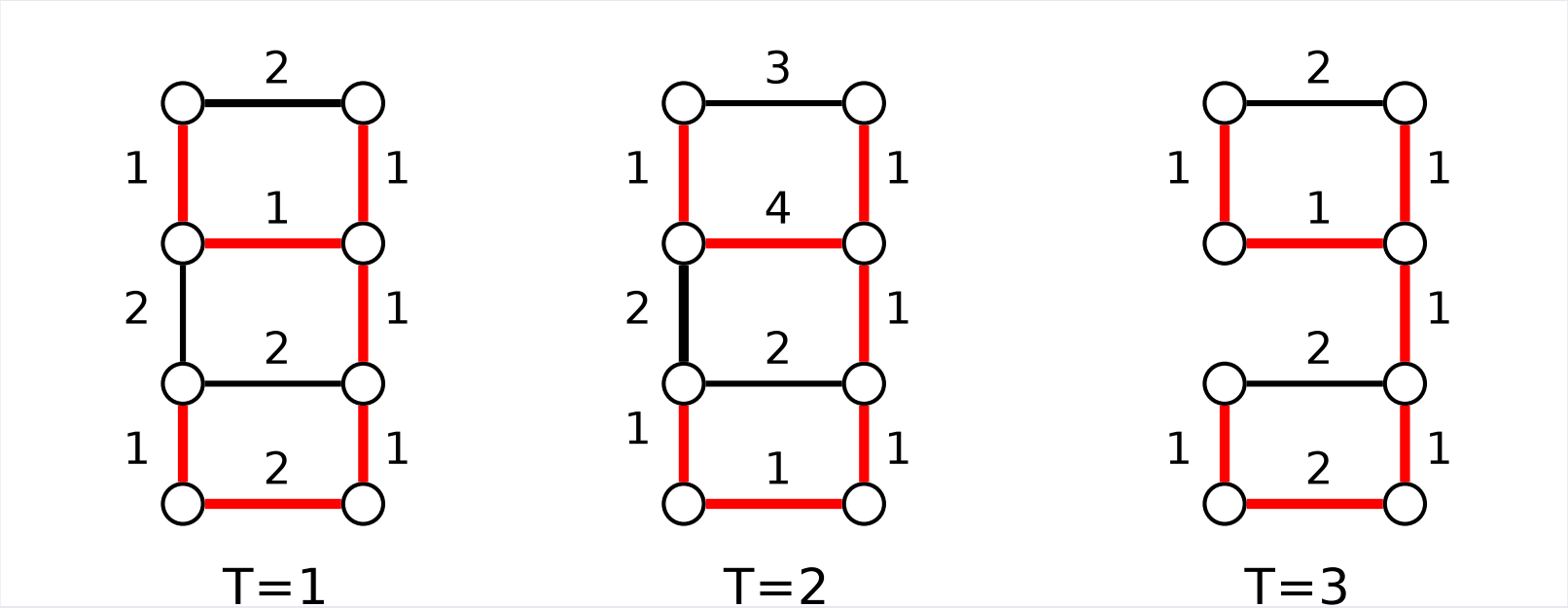
Input: T copies of the same graph, with different edge weights
Output: T spanning trees
Objective: Min weight cost + buying cost



Example: $T = 3$, buy$= 1$
Cost = 8+8+1+8+1=26

Temporal Spanning Trees

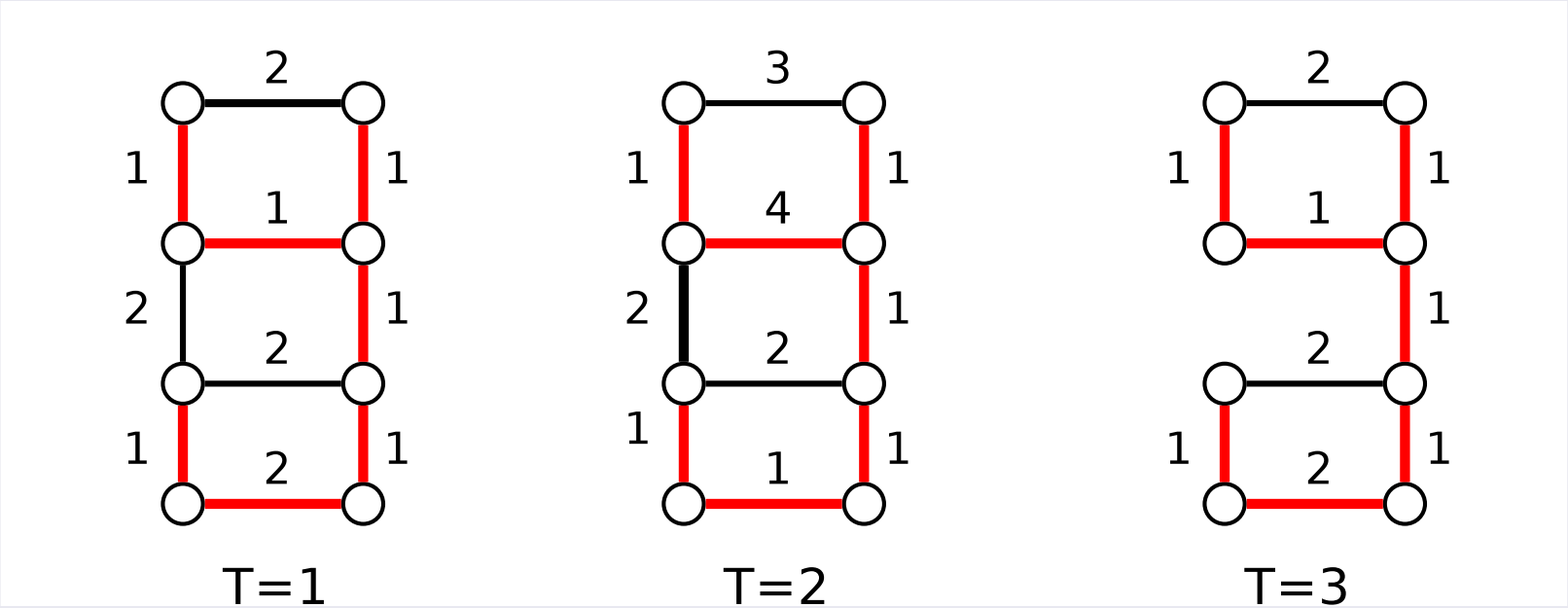
Input: T copies of the same graph, with different edge weights
Output: T spanning trees
Objective: Min weight cost + buying cost



Example: $T = 3$, buy$= 1$
Cost = 8+10+0+8+0=26

Temporal Spanning Trees

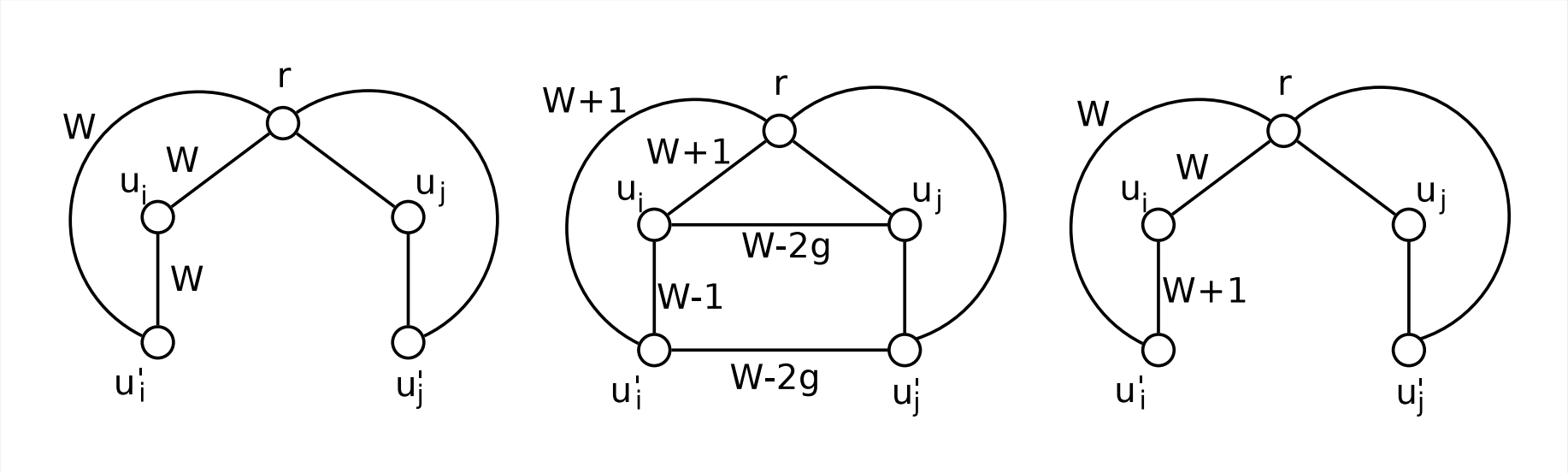
Input: T copies of the same graph, with different edge weights
Output: T spanning trees
Objective: Min weight cost + buying cost



Example: $T = 3$, buy$= 1$
Cost = 8+10+0+8+0=26

- In P when buy=0
- In P when buy=∞

Temporal Spanning Tree – APX-hardness

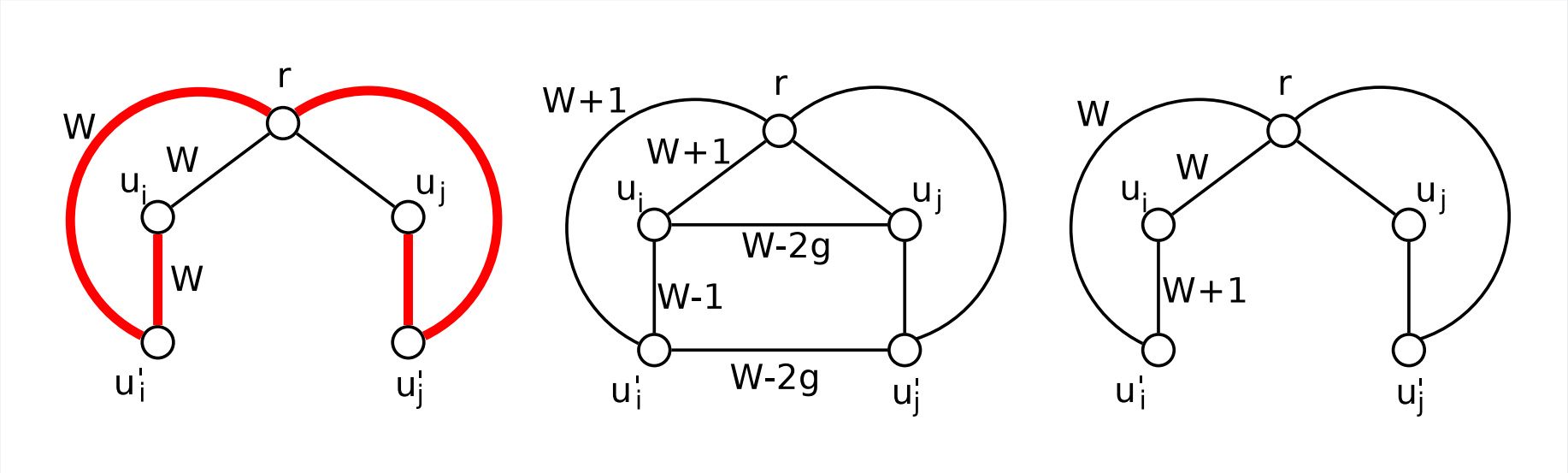


APX-hardness

Gadget that implements NAND constraint

Choices represented by “internal” edges used

Temporal Spanning Tree – APX-hardness



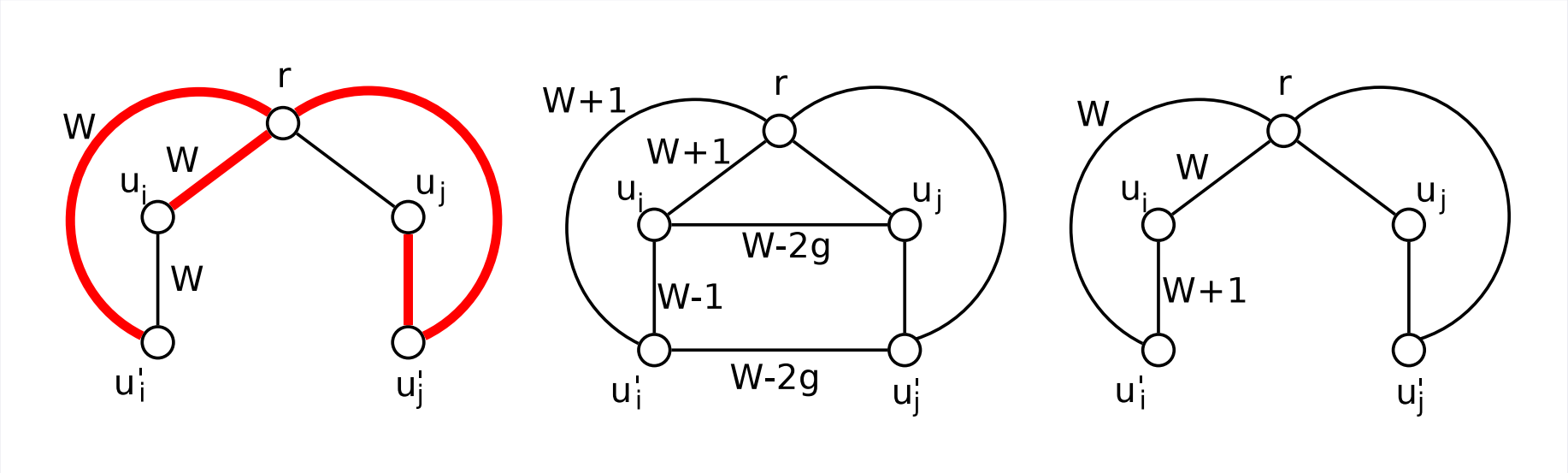
APX-hardness

Gadget that implements NAND constraint

Choices represented by “internal” edges used

Both used (BAD)

Temporal Spanning Tree – APX-hardness



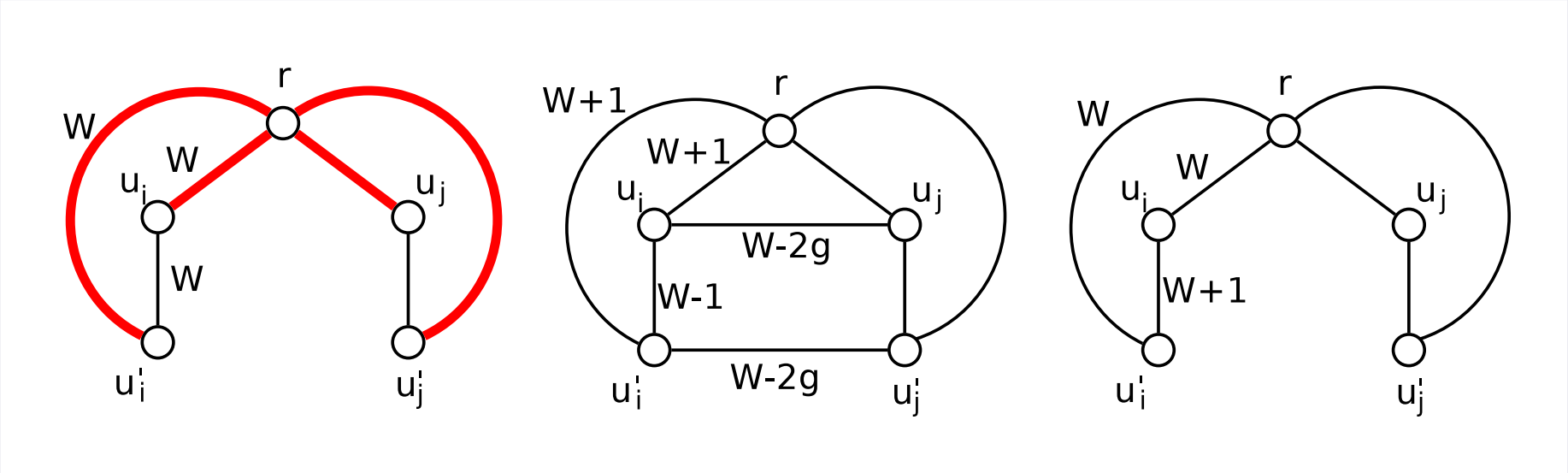
APX-hardness

Gadget that implements NAND constraint

Choices represented by “internal” edges used

One used (Good)

Temporal Spanning Tree – APX-hardness



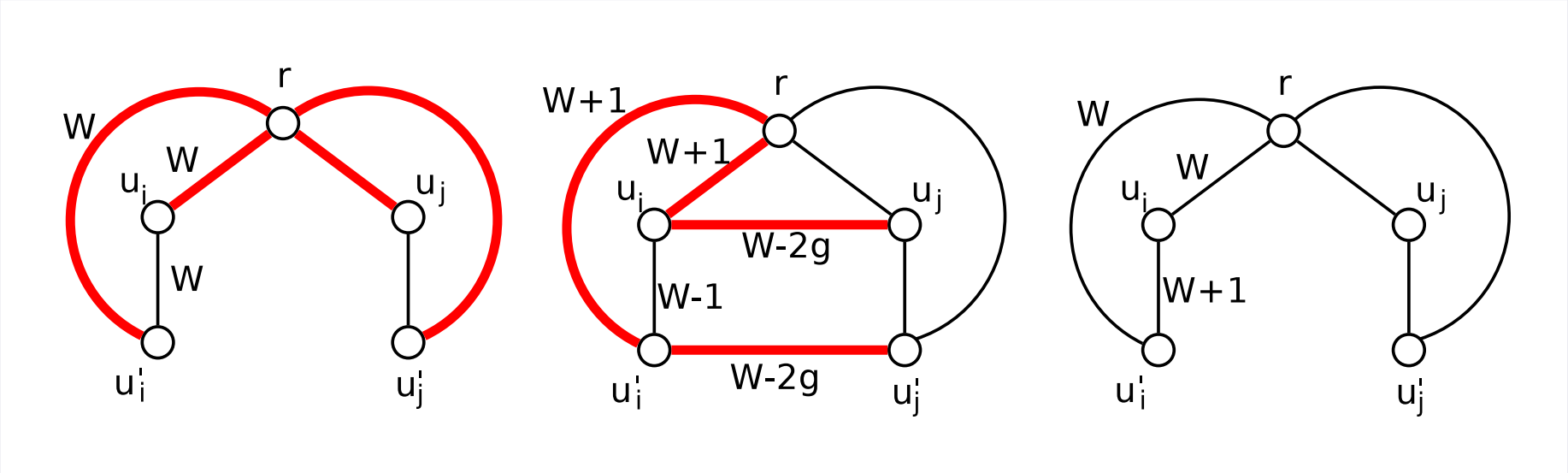
APX-hardness

Gadget that implements NAND constraint

Choices represented by “internal” edges used

None used (Good)

Temporal Spanning Tree – APX-hardness



APX-hardness

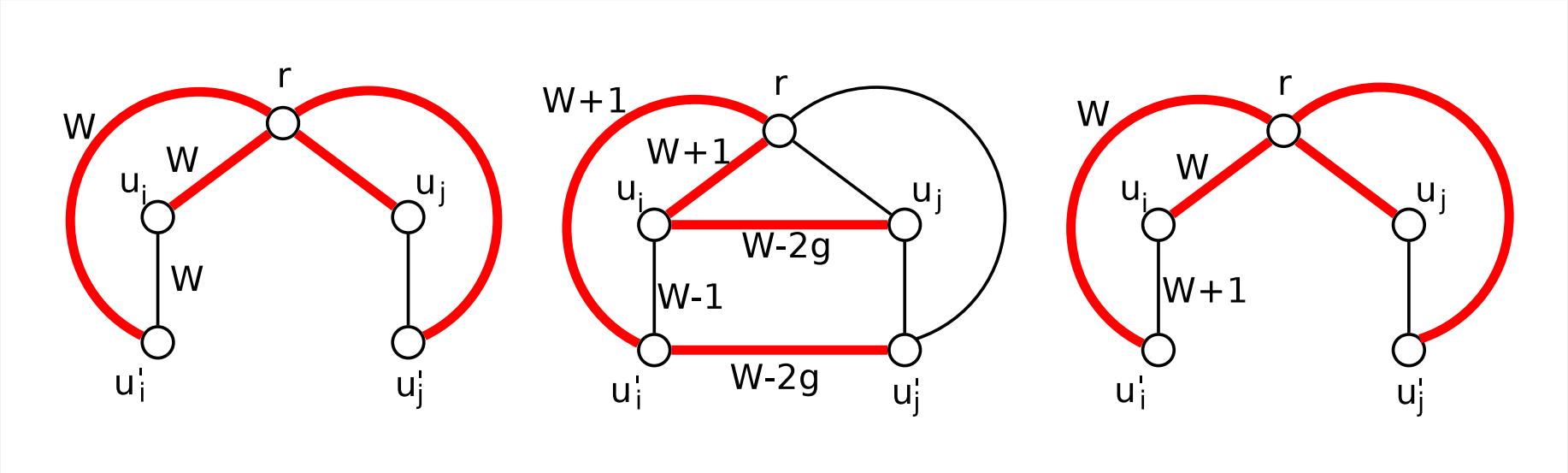
Gadget that implements NAND constraint

Choices represented by “internal” edges used

Goal: Replace edges from r with temporary cheap edges

Do this as much as possible

Temporal Spanning Tree – APX-hardness



APX-hardness

Gadget that implements NAND constraint

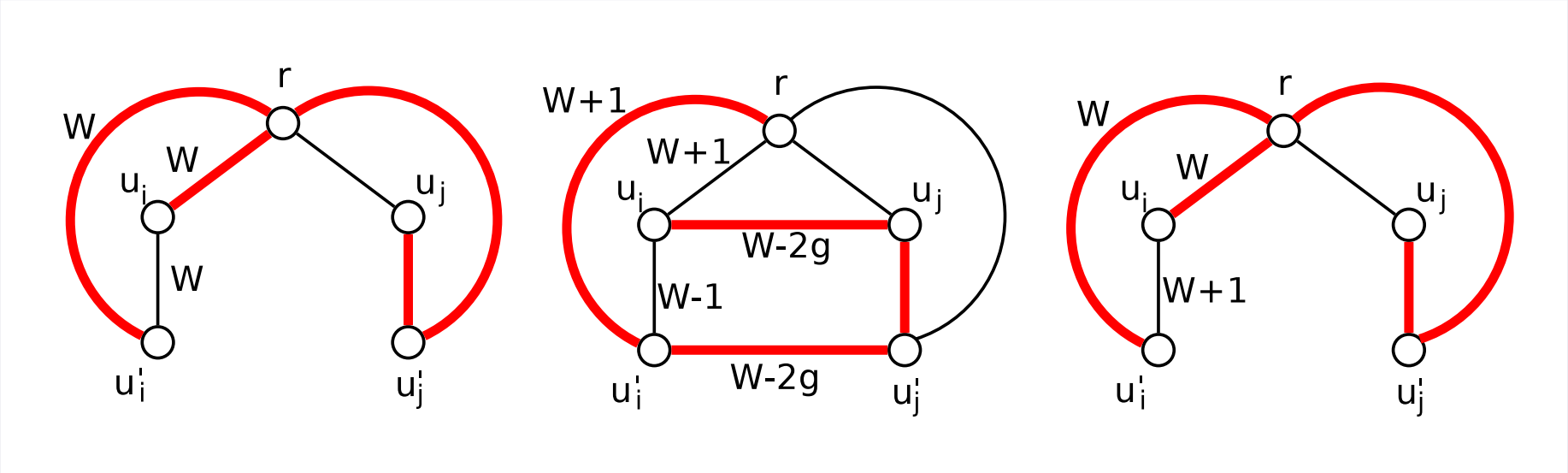
Choices represented by “internal” edges used

Goal: Replace edges from r with temporary cheap edges

Do this as much as possible

OK for no internal edges

Temporal Spanning Tree – APX-hardness



APX-hardness

Gadget that implements NAND constraint

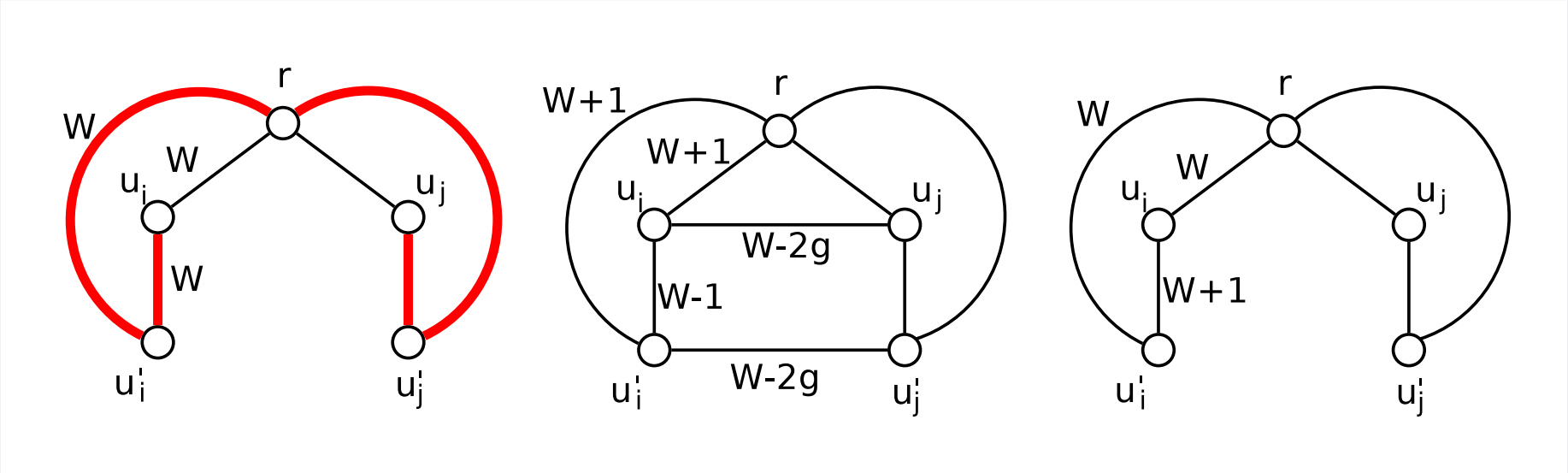
Choices represented by “internal” edges used

Goal: Replace edges from r with temporary cheap edges

Do this as much as possible

OK for one internal edge

Temporal Spanning Tree – APX-hardness



APX-hardness

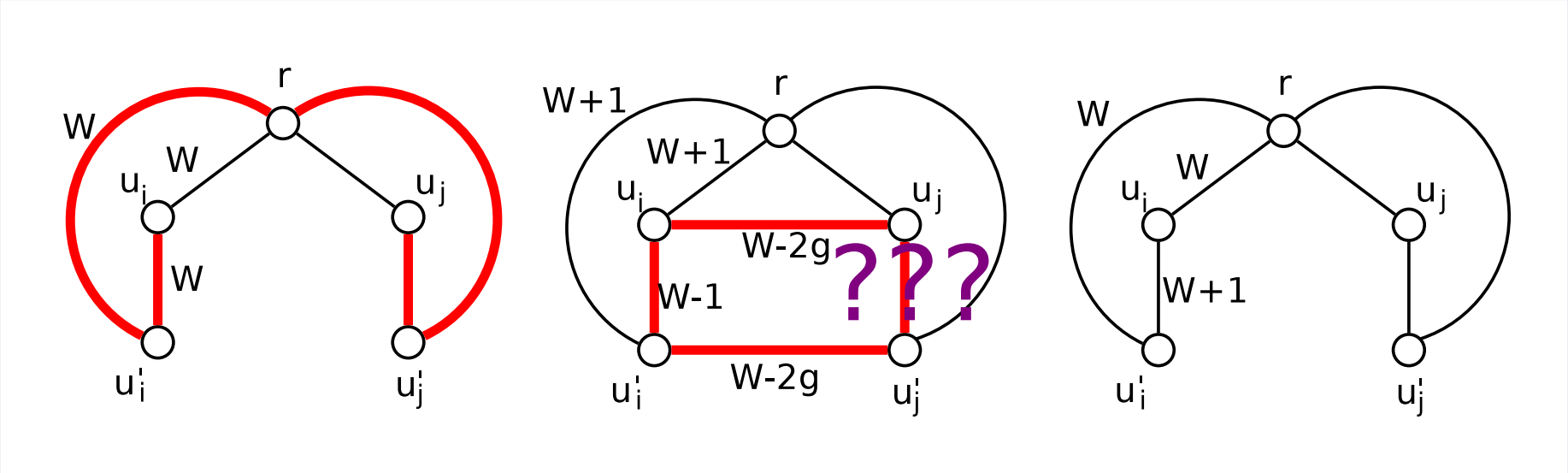
Gadget that implements NAND constraint

Choices represented by “internal” edges used

Goal: Replace edges from r with temporary cheap edges

Do this as much as possible

Temporal Spanning Tree – APX-hardness



APX-hardness

Gadget that implements NAND constraint

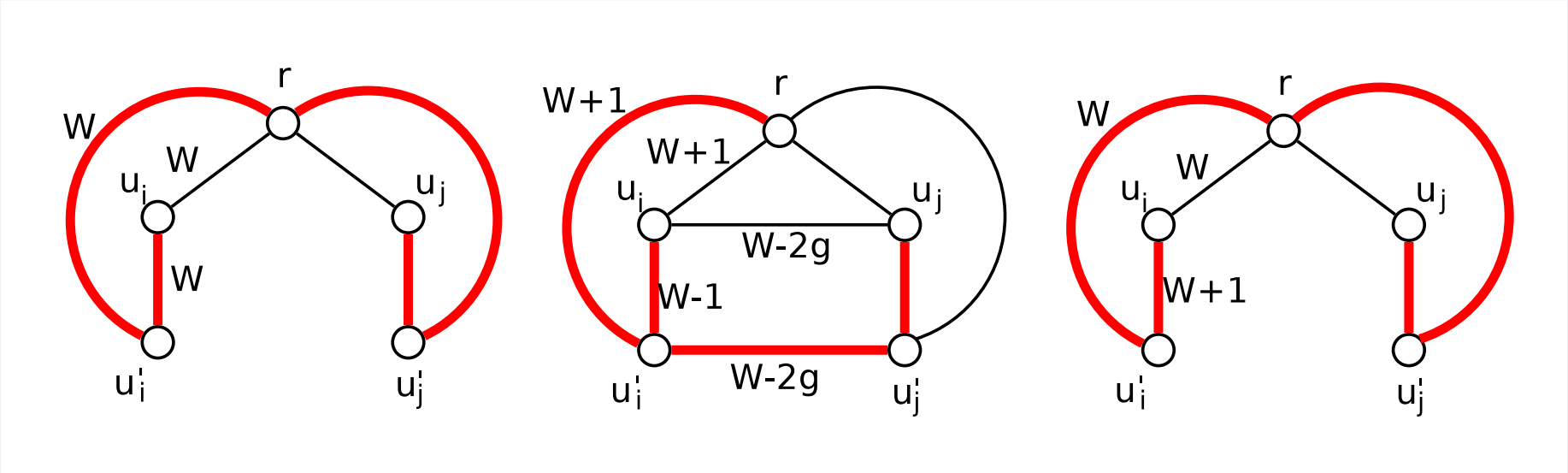
Choices represented by “internal” edges used

Goal: Replace edges from r with temporary cheap edges

Do this as much as possible

Not OK for two internal edges!

Temporal Spanning Tree – APX-hardness



APX-hardness

Gadget that implements NAND constraint

Choices represented by “internal” edges used

Goal: Replace edges from r with temporary cheap edges

Do this as much as possible

Not OK for two internal edges!

Temporal Spanning Tree – APX-hardness

APX-hardness

- Reduction from Independent Set
  - One “internal edge” for each vertex
  - One “testing phase” (NAND) for each edge
- Can make $T = O(1)$ by testing in parallel
- Reduction is linear

Theorem: There is no Approximation Scheme for TST in time $2^{n^{1-\delta}}$ (under ETH) even for constant rounds.

Temporal Spanning Tree – APX-hardness

APX-hardness

- Reduction from Independent Set
  - One “internal edge” for each vertex
  - One “testing phase” (NAND) for each edge
- Can make $T = O(1)$ by testing in parallel
- Reduction is linear

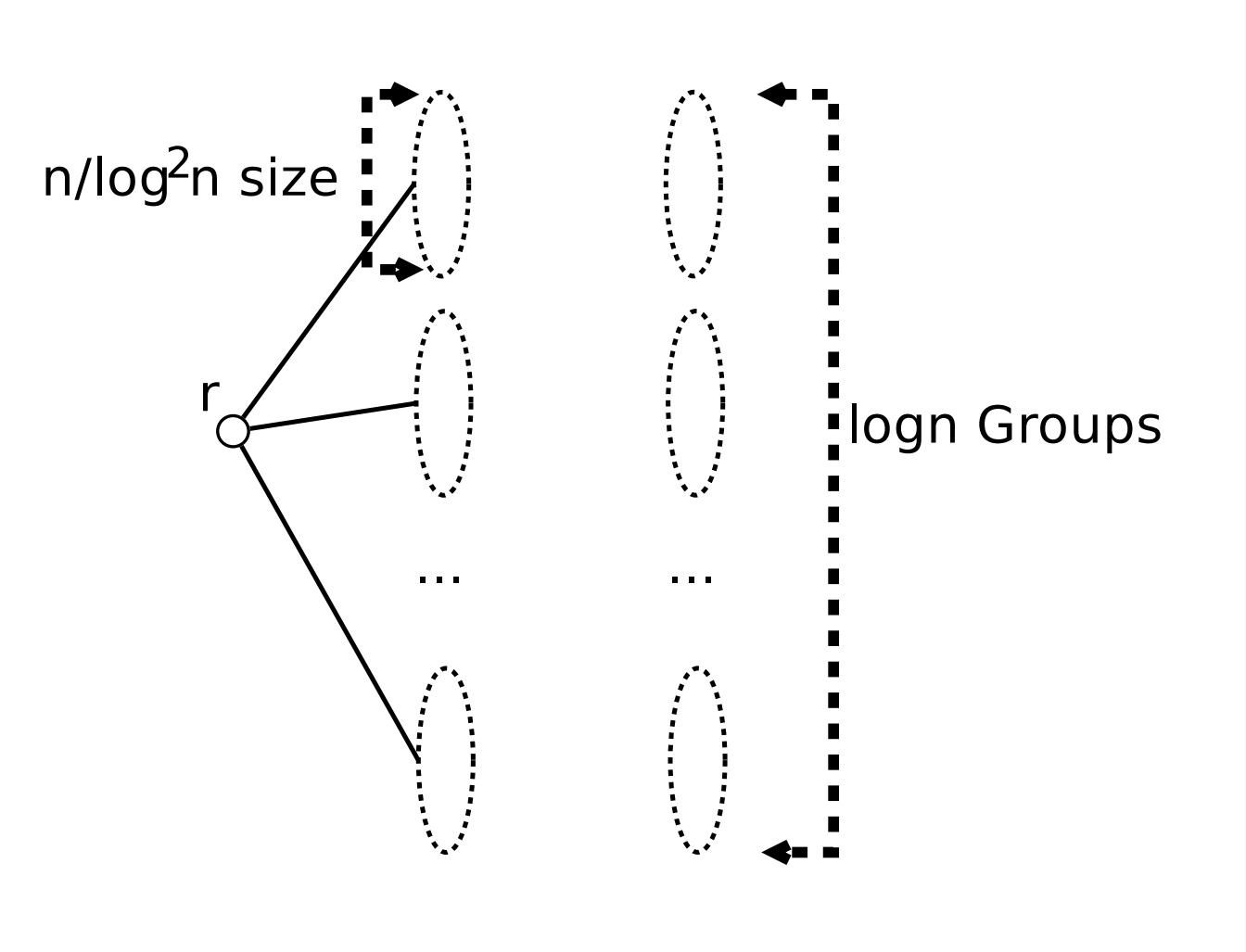
Theorem: There is no Approximation Scheme for TST in time $2^{n^{1-\delta}}$ (under ETH) even for constant rounds.

- For $T = 2$?
- Constant factor approximation algorithm?

Temporal Spanning Tree – Super-Exponential Lower Bound

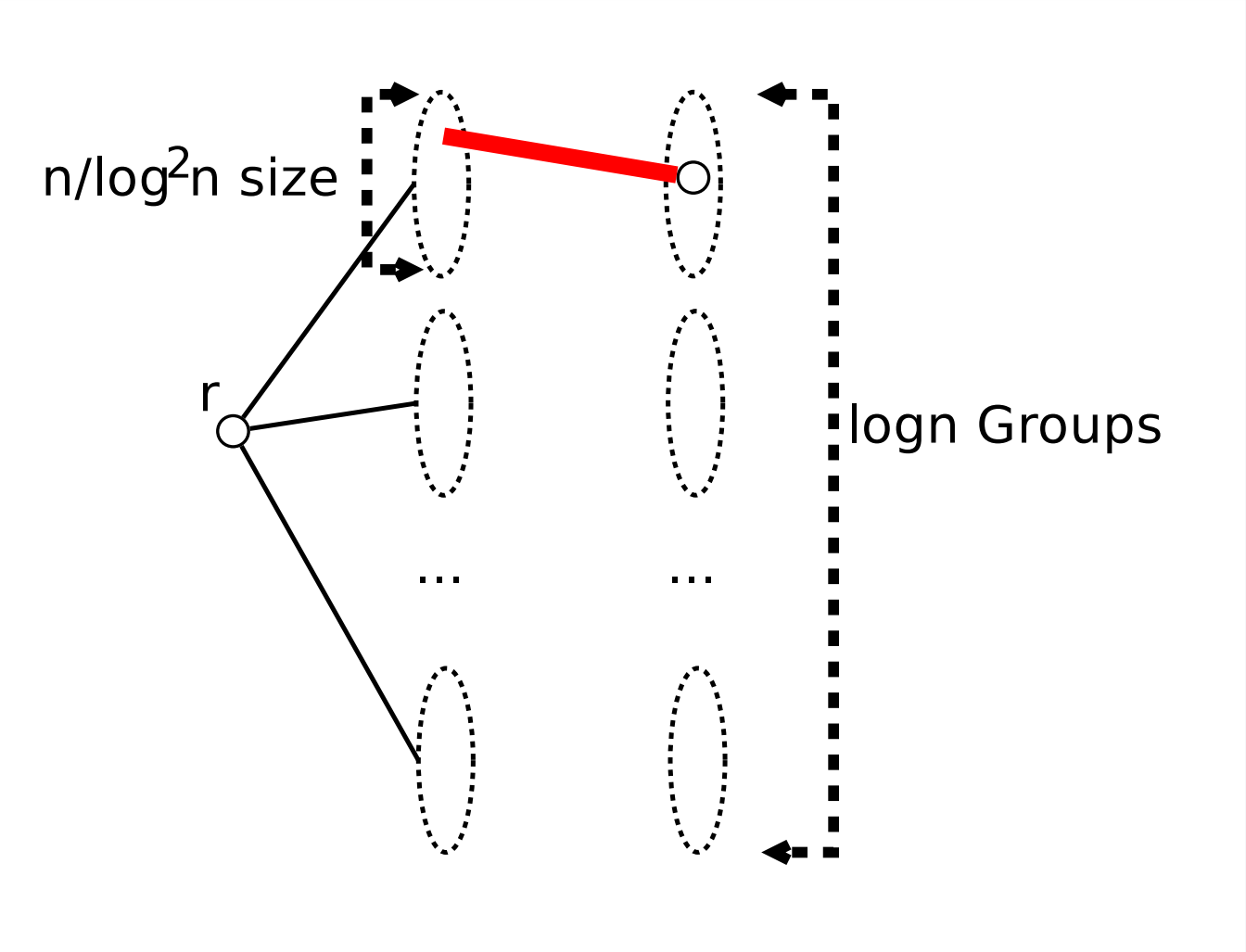
No talk is complete without a picture of a reduction that nobody understands (Daniel Marx)

Temporal Spanning Tree – Super-Exponential Lower Bound



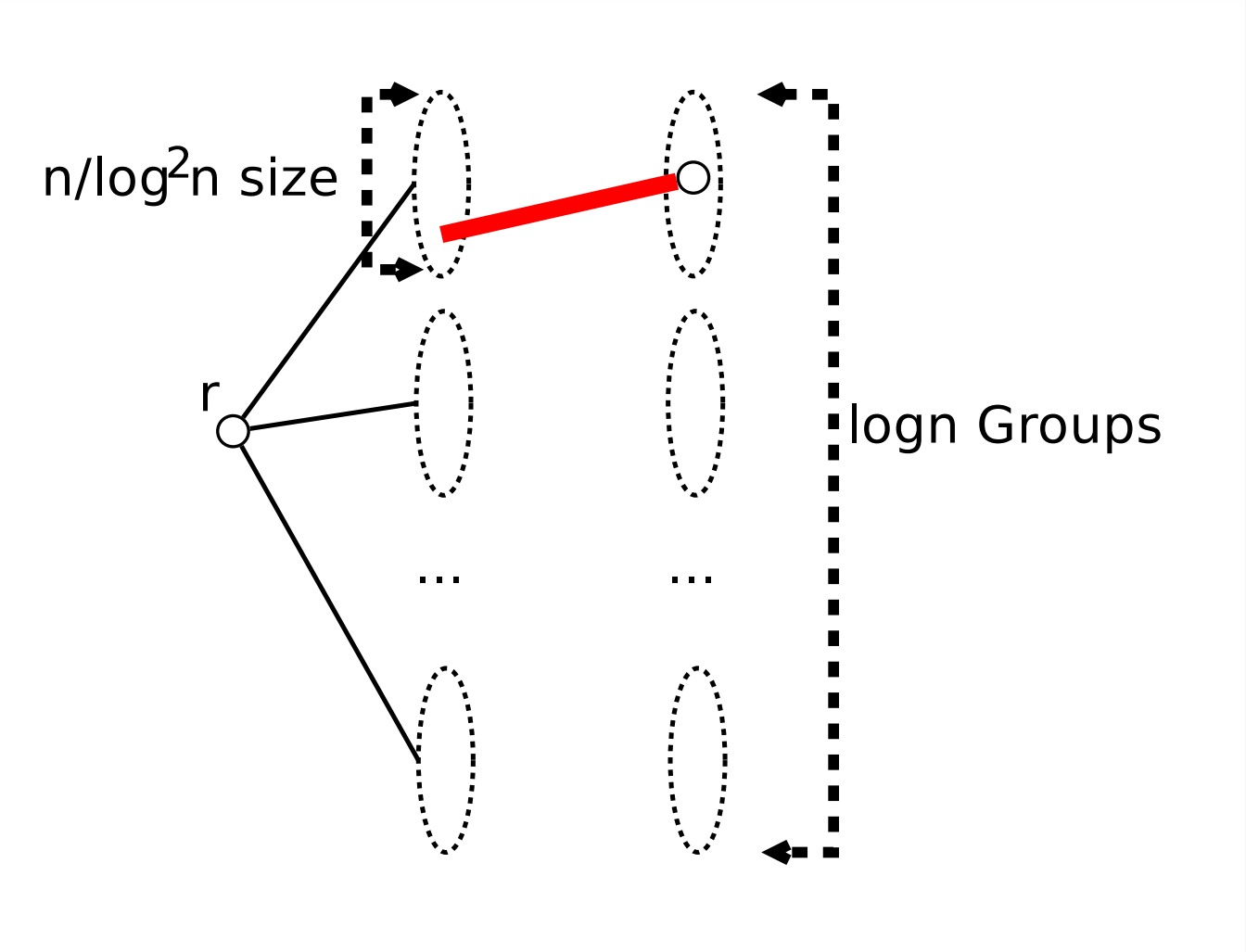
- Represent an n-var 3-SAT formula
- Edges used on right encode $\log n$ bits
- NAND gadget catches incosistent assignments
- Total order: $O(n/\log n)$

Temporal Spanning Tree – Super-Exponential Lower Bound



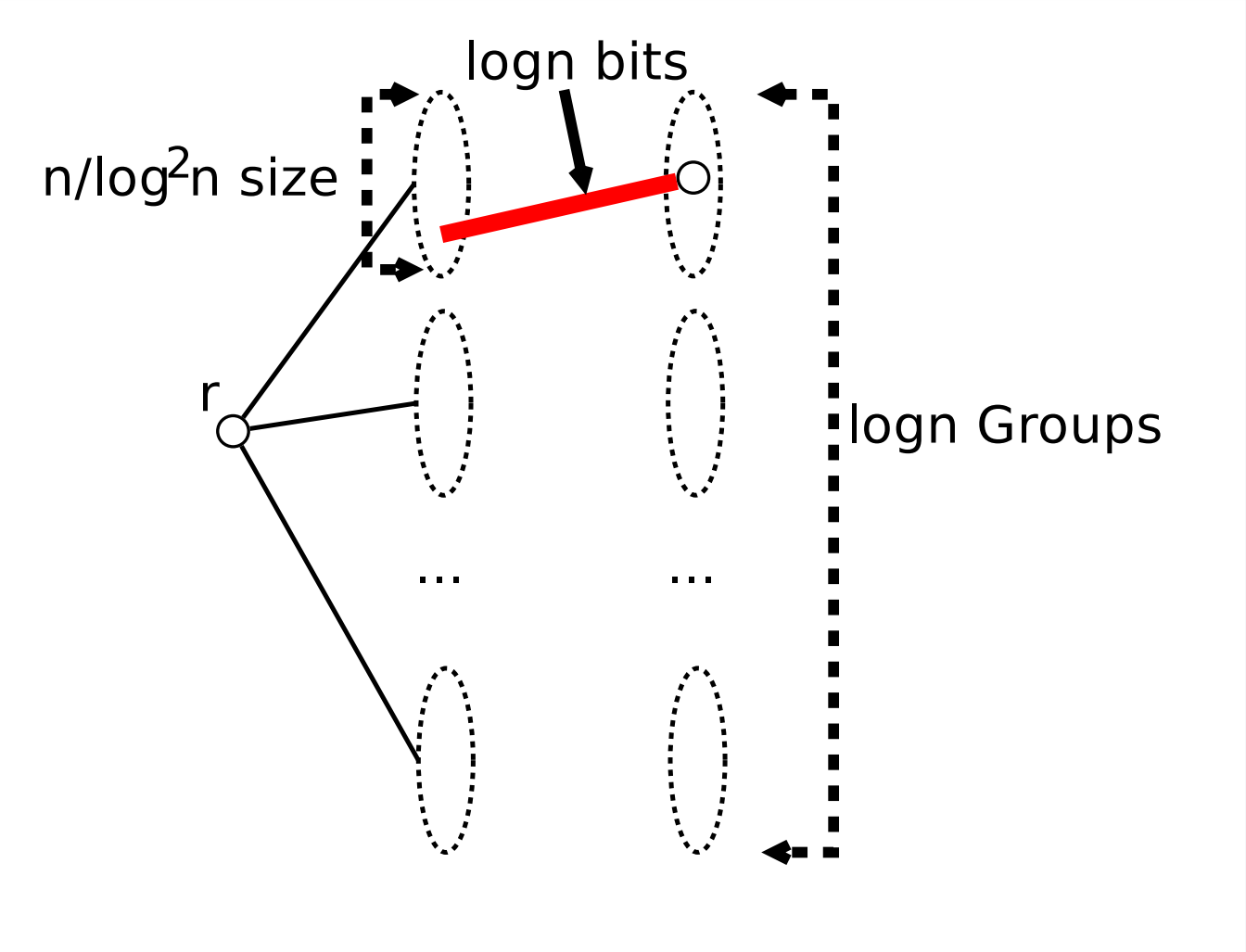
- Represent an n-var 3-SAT formula
- Edges used on right encode $\log n$ bits
- NAND gadget catches incosistent assignments
- Total order: $O(n/\log n)$

Temporal Spanning Tree – Super-Exponential Lower Bound



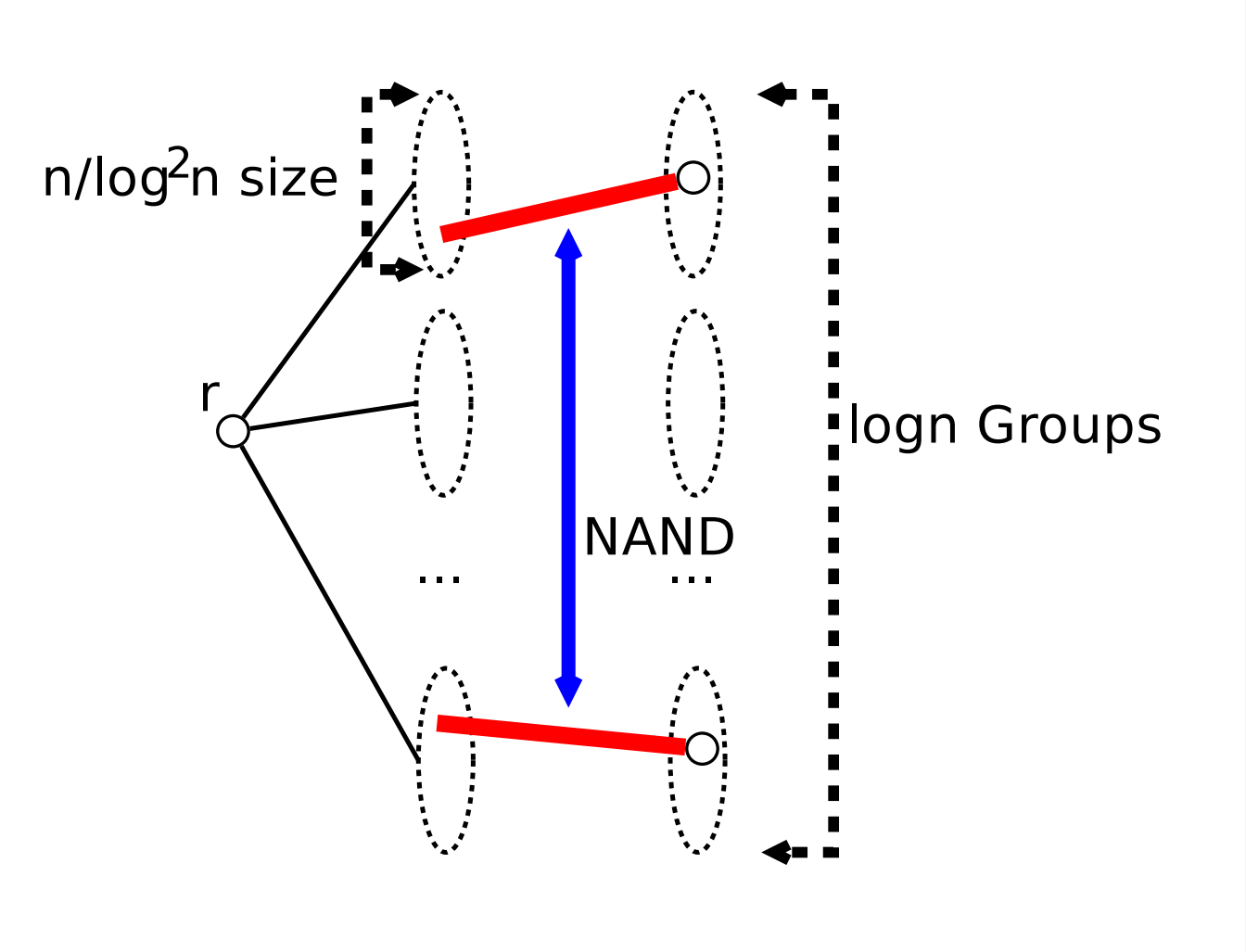
- Represent an n-var 3-SAT formula
- Edges used on right encode $\log n$ bits
- NAND gadget catches incosistent assignments
- Total order: $O(n/\log n)$

Temporal Spanning Tree – Super-Exponential Lower Bound



- Represent an n-var 3-SAT formula
- Edges used on right encode $\log n$ bits
- NAND gadget catches incosistent assignments
- Total order: $O(n/\log n)$

Temporal Spanning Tree – Super-Exponential Lower Bound



- Represent an n-var 3-SAT formula
- Edges used on right encode $\log n$ bits
- NAND gadget catches incosistent assignments
- Total order: $O(n/\log n)$

Temporal Spanning Tree – Super-Exponential Lower Bound

- Reduction 3-SAT $\rightarrow$ TST
  - Formula with n vars
  - New graph has $n/\log n$ vertices
  - If $n^{o(n)}$ alg for TST...
  - ...then $2^{o(n)}$ alg for 3-SAT
- TST needs super-exponential time (n^n) to be solved! (under ETH)
- $n^n T$ agorithm is easy...(DP, list all spanning trees)

Temporal Spanning Tree – Super-Exponential Lower Bound

- Reduction 3-SAT $\rightarrow$ TST
  - Formula with n vars
  - New graph has $n/\log n$ vertices
  - If $n^{o(n)}$ alg for TST...
  - ...then $2^{o(n)}$ alg for 3-SAT
- TST needs super-exponential time (n^n) to be solved! (under ETH)
- $n^n T$ agorithm is easy...(DP, list all spanning trees)

What does this have to do with anything?

Temporal Spanning Tree – Few Rounds

- TST needs n^n time to solve exactly.
- Can we do c^n if $T = O(1)$?

Temporal Spanning Tree – Few Rounds

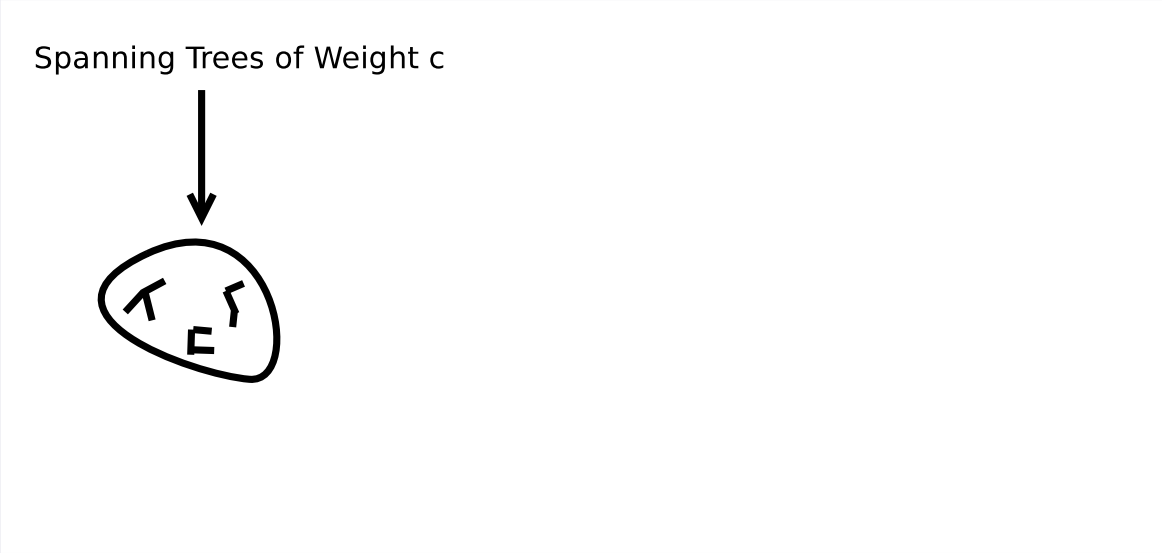
- TST needs n^n time to solve exactly.
- Can we do c^n if $T = O(1)$?

The Cut&Count method

Temporal Spanning Tree – Few Rounds

- TST needs n^n time to solve exactly.
- Can we do c^n if $T = O(1)$?

The Cut&Count method

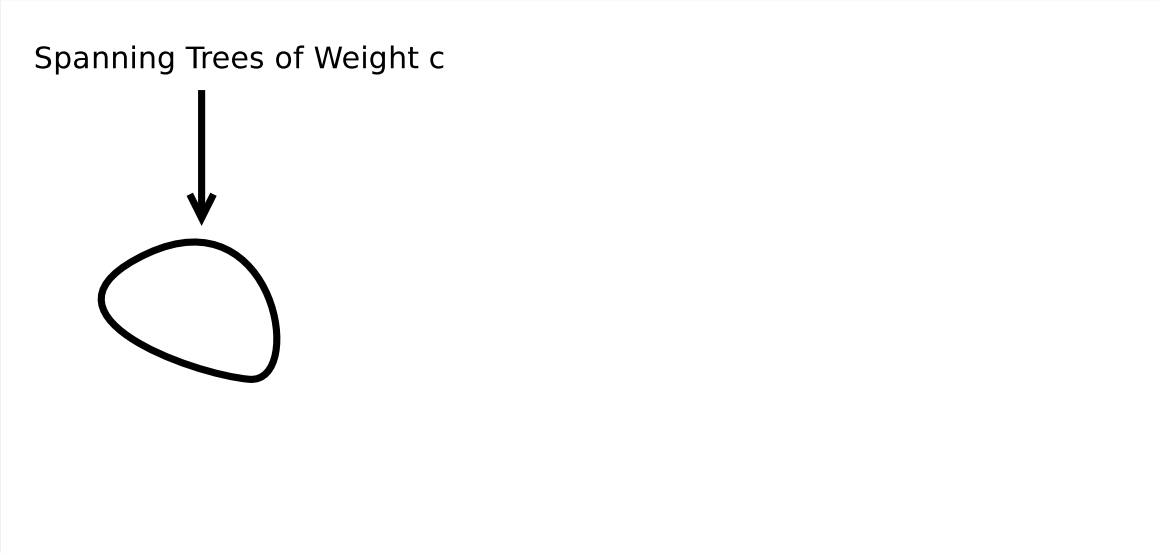


Consider the set of STs of weight c

Temporal Spanning Tree – Few Rounds

- TST needs n^n time to solve exactly.
- Can we do c^n if $T = O(1)$?

The Cut&Count method

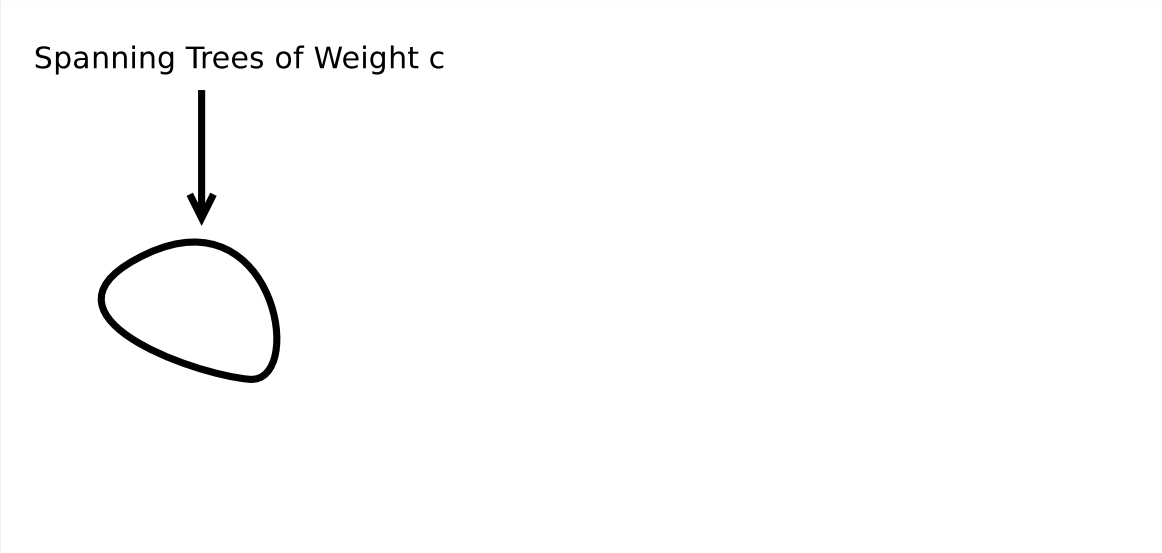


Consider the set of STs of weight c
Is this set empty?

Temporal Spanning Tree – Few Rounds

- TST needs n^n time to solve exactly.
- Can we do c^n if $T = O(1)$?

The Cut&Count method



Consider the set of STs of weight c
Is this set empty?
We will decide by determining its parity!!!

Temporal Spanning Tree – Few Rounds

- TST needs n^n time to solve exactly.
- Can we do c^n if $T = O(1)$?

The Cut&Count method
Two problems:

- Determining the parity is potentially harder
- What if the # of solutions is even?

Temporal Spanning Tree – Few Rounds

- TST needs n^n time to solve exactly.
- Can we do c^n if $T = O(1)$?

The Cut&Count method
Two problems:

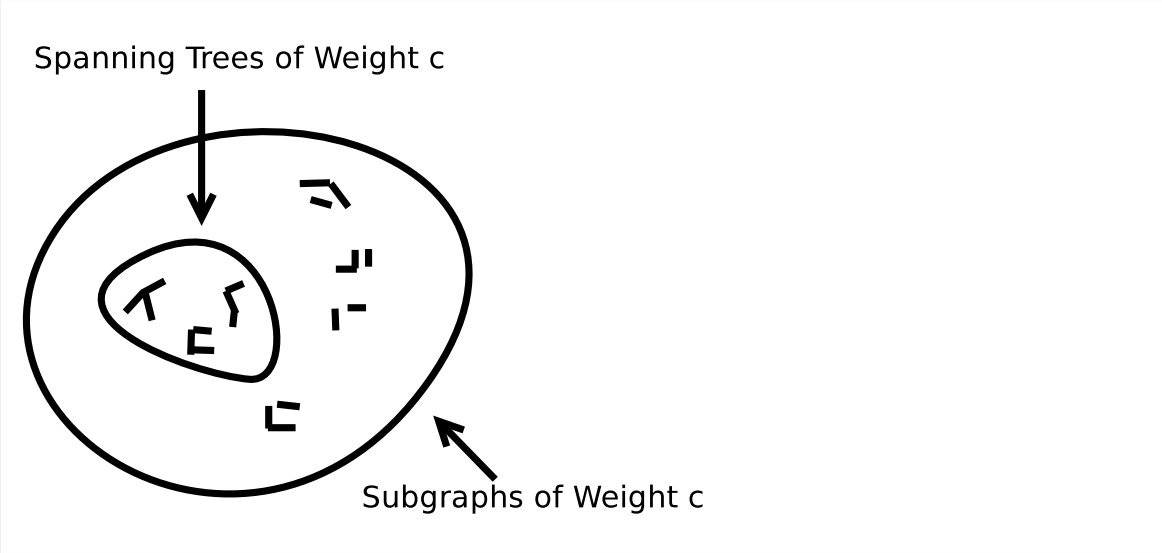
- Determining the parity is potentially harder
- What if the # of solutions is even?

- Bad solutions cancel out! $\rightarrow$ we can do this in c^n.
- Isolation Lemma $\rightarrow$ minimum solution is unique whp.

Temporal Spanning Tree – Few Rounds

- TST needs n^n time to solve exactly.
- Can we do c^n if $T = O(1)$?

The Cut&Count method

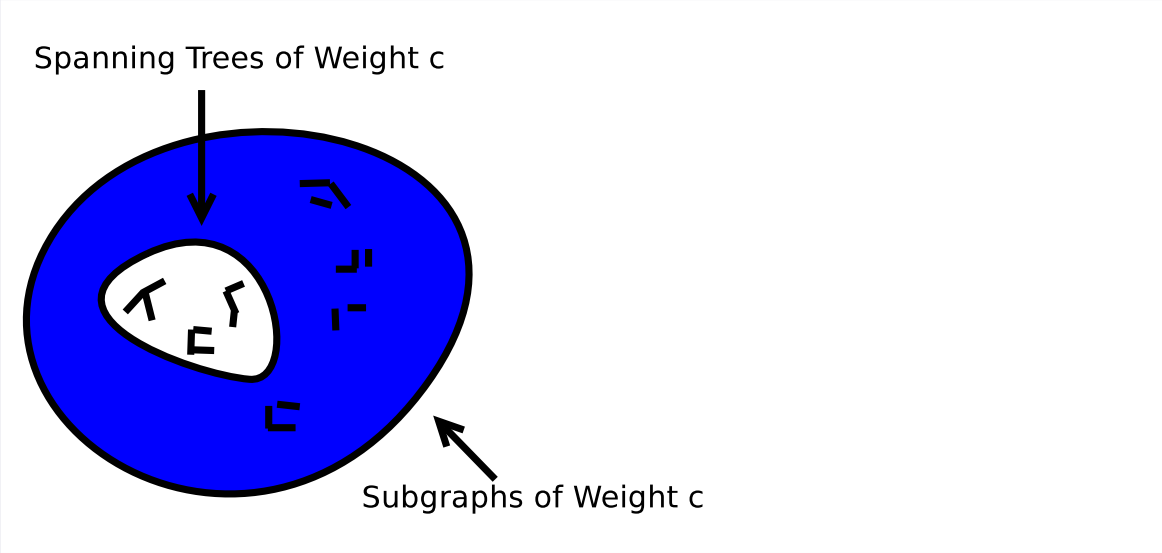


- We can determine the size of bigger set in 2^n time (DP)

Temporal Spanning Tree – Few Rounds

- TST needs n^n time to solve exactly.
- Can we do c^n if $T = O(1)$?

The Cut&Count method

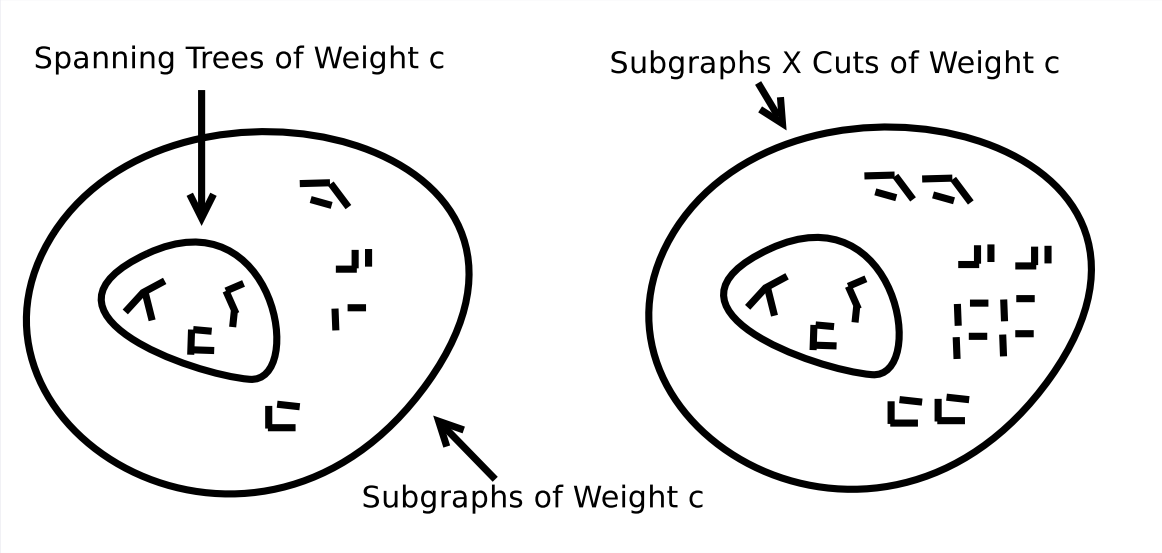


- We can determine the size of bigger set in 2^n time (DP)
- If blue area has even size $\rightarrow$ Done!

Temporal Spanning Tree – Few Rounds

- TST needs n^n time to solve exactly.
- Can we do c^n if $T = O(1)$?

The Cut&Count method



- We can determine the size of bigger set in 2^n time (DP)
- Construct a new set:
  - Every subgraph appears one time for each cut of the graph it respects.
  - Good subgraphs only respect one cut! (the trivial one)

Temporal Spanning Tree – An Approximation Scheme

- TST is APX-hard for time $2^{n^{1-\delta}}$.
- TST needs n^n time to solve exactly.
- Can we $(1+\epsilon)$-approximate in single-exponential time? (c^n)

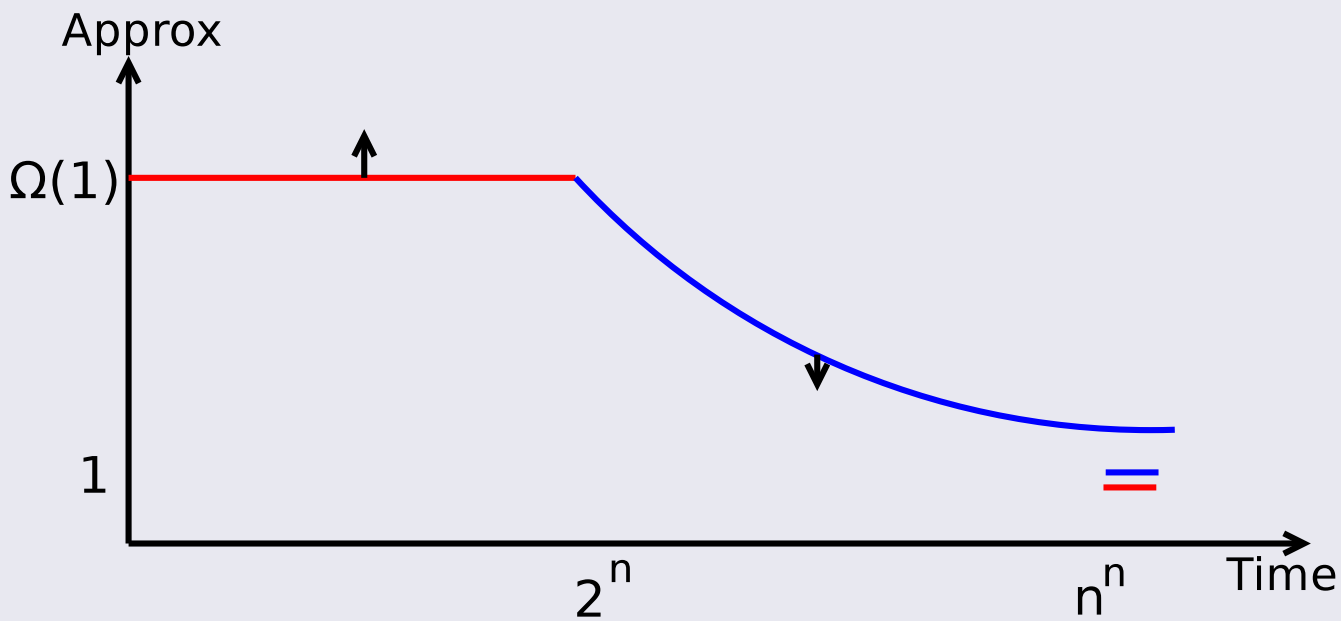
Temporal Spanning Tree – An Approximation Scheme

- TST is APX-hard for time $2^{n^{1-\delta}}$.
- TST needs n^n time to solve exactly.
- Can we $(1+\epsilon)$-approximate in single-exponential time? (c^n)

- If $T = O(1)$ Yes! (previous slide)
- Otherwise, divide T into blocks of size $1/\epsilon$
- Solve each optimally
  - Added buying cost is at most $\epsilon \mathrm{OPT}$
- $(1+\epsilon)$-approximation in time $2^{O(n/\epsilon)}$.

- Is this trade-off optimal??

Temporal Spanning Tree – Summary



- At least constant inapproximability in $2^{o(n)}$ time.
- Any $(1+\epsilon)$ approximation in $2^{n/\epsilon}$ time.
- n^n time (tight) to solve exactly.

Temporal Spanning Tree – Summary

- At least constant inapproximability in $2^{o(n)}$ time.
- Any $(1+\epsilon)$ approximation in $2^{n/\epsilon}$ time.
- n^n time (tight) to solve exactly.

Approximation helps for fighting huge but soft lower bounds

Conclusions

- Time doesn't buy approximation
  - Except for very hard problems...
- Time buys generality
  - Time vs. Generality vs. Approximation trade-offs?
  - A 3-D theory of approximation??
- Use approximation to fight ETH/SETH lower bounds?
  - Double-exponential bounds? (see Valia's talk tomorrow)
  - SETH bounds? E.g. $(2-\epsilon)^{tw}$ for VC?

Thank you!