Exponential-Time Approximation of Hard Problems

Łukasz Kowalik
joint work with: Marek Cygan, Marcin Pilipczuk and Mateusz Wykurz

University of Warsaw, Poland

Dahstuhl Seminar on Moderately Exponential Time Algorithms,
19-24.10.2008

Some NP-hard problems are really hard

We will focus on the following, natural problems:

- Set Cover
- Bandwidth
- Vertex Coloring
- Maximum Independent Set

Coping with NP-hardness

1. (poly-time) approximation.

Coping with NP-hardness

1. (poly-time) approximation.
   - SET COVER: no $(1-\epsilon)\log n$-approximation, unless $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{\log\log n})$.
   - BANDWIDTH: no $O(1)$-approximation, unless $NP = P$
   - VERTEX COLORING: no $n^{1-\epsilon}$-approximation, unless $NP = ZPP$
   - MAXIMUM INDEPENDENT SET: no $n^{1-\epsilon}$-approximation, unless $NP = ZPP$

Coping with NP-hardness

1. (poly-time) approximation.
2. Fixed-parameter tractability

Coping with NP-hardness

1. (poly-time) approximation.
2. Fixed-parameter tractability
   - SET COVER: $W[2]$-complete.
   - BANDWIDTH: $W[t]$-hard, for any $t > 0$.
   - k-COLORING: NP-complete for any $k \geq 3$.
   - MAXIMUM INDEPENDENT SET: $W[1]$-complete

Coping with NP-hardness

1. (poly-time) approximation.
2. Fixed-parameter tractability
3. Moderately exponential-time exact algorithms

Coping with NP-hardness

1. (poly-time) approximation.
2. Fixed-parameter tractability
3. Moderately exponential-time exact algorithms
   - Set Cover: $O^*(2^m)$, $O^*(4^n)$, $O^*(2^{0.299(n+m)})$.
   - Bandwidth: $O^*(5^n)$-time and $O^*(2^n)$-space; $O^*(10^n)$ poly-space,.
   - k-coloring: $O^*(2^n)$-time and space.
   - Maximum Independent Set: $O(2^{0.276n})$-time, exp-space; $O(2^{0.288n})$-time, poly-space.

Coping with NP-hardness

1. (poly-time) approximation.
2. Fixed-parameter tractability
3. Moderately exponential-time exact algorithms
4. Moderately exponential-time approximation algorithms (our approach)

Approach One:
Reducing the Instance Size

Let us recall the UNWEIGHTED SET COVER problem:

Instance

Collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$

The union $\bigcup \mathcal{S}$ is called the universe and denoted by U.

Problem

Find the smallest possible subcollection $\mathcal{C} \subseteq \mathcal{S}$ so that $\bigcup \mathcal{C} = U$.

Unweighted Set Cover, reducing the number of sets

Approximation algorithm:

1. Join the sets of $\mathcal{S}$ into pairs:
   $S'_i = S_{2i-1} \cup S_{2i}$, for $i = 1, \ldots, m/2$ (assume m even),
   Create new instance $\mathcal{S}' = \{S'_i \mid i = 1, \ldots, m/2\}$.
2. Solve the problem for instance $\mathcal{S}'$ by the exact algorithm, in time $O(2^{m/2})$. Let $\mathcal{C}'$ be the solution.
3. Transform $\mathcal{C}'$ into a cover of $\mathcal{S}$: $\mathcal{C} = \{S_{2i-1} \cup S_{2i} \mid S'_i \in \mathcal{C}'\}$.

Unweighted Set Cover, reducing the number of sets

Approximation algorithm:

1. Join the sets of $\mathcal{S}$ into pairs:
   $S'_i = S_{2i-1} \cup S_{2i}$, for $i = 1, \ldots, m/2$ (assume m even),
   Create new instance $\mathcal{S}' = \{S'_i \mid i = 1, \ldots, m/2\}$.
2. Solve the problem for instance $\mathcal{S}'$ by the exact algorithm, in time $O(2^{m/2})$. Let $\mathcal{C}'$ be the solution.
3. Transform $\mathcal{C}'$ into a cover of $\mathcal{S}$: $\mathcal{C} = \{S_{2i-1} \cup S_{2i} \mid S'_i \in \mathcal{C}'\}$.

Proposition

This is a 2-approximation

Proof.

Let OPT be the size of the optimal cover for $\mathcal{S}$. In $\mathcal{S}'$ there is a cover of size $\leq \mathrm{OPT}$ Hence $|\mathcal{C}'| \leq \mathrm{OPT}$ and $|\mathcal{C}| \leq 2\mathrm{OPT}$. □

Approximation algorithm:

1. Join the sets of $\mathcal{S}$ into pairs:
   $S'_i = S_{2i-1} \cup S_{2i}$, for $i = 1, \ldots, m/2$ (assume m even),
   Create new instance $\mathcal{S}' = \{S'_i \mid i = 1, \ldots, m/2\}$.
2. Solve the problem for instance $\mathcal{S}'$ by the exact algorithm, in time $O(2^{m/2})$. Let $\mathcal{C}'$ be the solution.
3. Transform $\mathcal{C}'$ into a cover of $\mathcal{S}$: $\mathcal{C} = \{S_{2i-1} \cup S_{2i} \mid S'_i \in \mathcal{C}'\}$.

Question

Does it work for the weighted case?

Approximation algorithm:

1. Join the sets of $\mathcal{S}$ into pairs:
   $S'_i = S_{2i-1} \cup S_{2i}$, for $i = 1, \ldots, m/2$ (assume m even),
   Create new instance $\mathcal{S}' = \{S'_i \mid i = 1, \ldots, m/2\}$.
2. Solve the problem for instance $\mathcal{S}'$ by the exact algorithm, in time $O(2^{m/2})$. Let $\mathcal{C}'$ be the solution.
3. Transform $\mathcal{C}'$ into a cover of $\mathcal{S}$: $\mathcal{C} = \{S_{2i-1} \cup S_{2i} \mid S'_i \in \mathcal{C}'\}$.

Question

Does it work for the weighted case?

Answer

Not quite: light sets from OPT may join with heavy sets. Sorting sets ???

Weighted Set Cover, reducing the number of sets

$$S_1 \le S_2 \le S_3 \le S_4 \le S_5 \le S_6 \le S_7 \le S_8 \le S_9 \le S_{10} \le S_{11} \le S_{12}$$

WEIGHTED SET COVER, reducing the number of sets

$$\boxed{S_1 \leq S_2} \leq \boxed{S_3 \leq S_4} \leq \boxed{S_5 \leq S_6} \leq \boxed{S_7 \leq S_8} \leq \boxed{S_9 \leq S_{10}} \leq \boxed{S_{11} \leq S_{12}}$$

The sets from optimal solution are marked green.

$$(S_1 \leq S_2) \leq (S_3 \leq S_4) \leq (S_5 \leq S_6) \leq (S_7 \leq S_8) \leq (S_9 \leq S_{10}) \leq (S_{11} \leq S_{12})$$

WEIGHTED SET COVER, reducing the number of sets

The sets from optimal solution are marked green.

$$(S_1 \le S_2) \le (S_3 \le S_4) \le (S_5 \le S_6) \le (S_7 \le S_8) \le (S_9 \le S_{10}) \le (S_{11} \le S_{12})$$

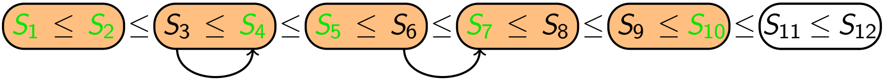
Weighted Set Cover, reducing the number of sets

The sets from optimal solution are marked green.

$$(S_1 \le S_2) \le (S_3 \le S_4) \le (S_5 \le S_6) \le (S_7 \le S_8) \le (S_9 \le S_{10}) \le (S_{11} \le S_{12})$$

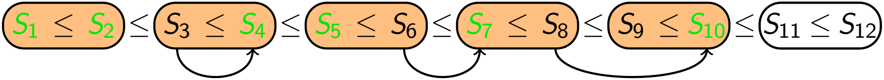
WEIGHTED SET COVER, reducing the number of sets

The sets from optimal solution are marked green.



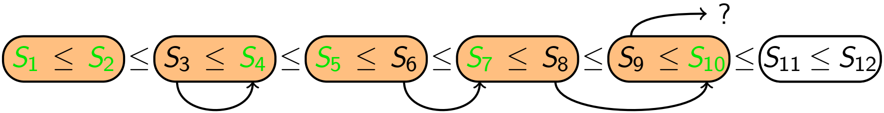
WEIGHTED SET COVER, reducing the number of sets

The sets from optimal solution are marked green.

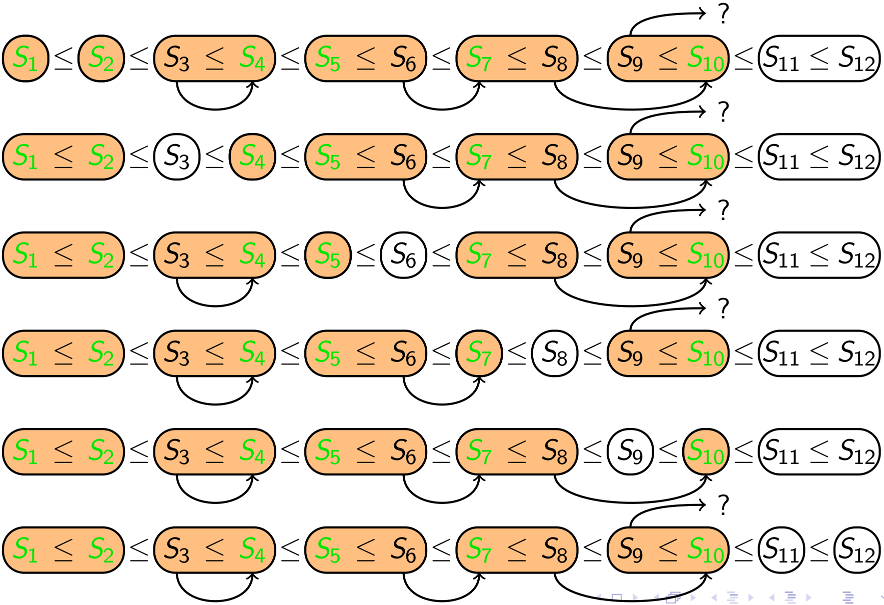


WEIGHTED SET COVER, reducing the number of sets

The sets from optimal solution are marked green.



WEIGHTED SET COVER, reducing the number of sets



WEIGHTED SET COVER, summary

Assume we have an exact $T(n)$-time algorithm for SET COVER.

- For any $r \in \mathbb{N}$ we have r-approximation in $m \cdot T(n/r)$ time (We have just seen it for $r = 2$),
- For any $r \in \mathbb{Q}$ we have $(\ln r + 1)$-approximation in $m \cdot T(n/r)$ time (We have seen it yesterday for unweighted version, for weighted version again it requires additional trick),

Example 2: Set Cover, reducing the universe

Recall the standard greedy $O(\log n)$-approximation algorithm:

Greedy

1: $\mathcal{C} \leftarrow \emptyset$.
2: **while** $\mathcal{C}$ does not cover U **do**
3: Find $T \in \mathcal{S}$ so as to minimize $\frac{w(T)}{|T \setminus \bigcup \mathcal{C}|}$
4: $\mathcal{C} \leftarrow \mathcal{C} \cup \{T\}$.

Example 2: SET COVER, reducing the universe

Recall the standard greedy $O(\log n)$-approximation algorithm:

Greedy

$\mathcal{C} \leftarrow \emptyset$.
while $\mathcal{C}$ does not cover U **do**
    Find $T \in \mathcal{S}$ so as to minimize $\frac{w(T)}{|T \setminus \bigcup \mathcal{C}|}$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{T\}$.
    for each $e \in T \setminus \bigcup \mathcal{C}$ **do**
        $\mathrm{price}(e) \leftarrow \frac{w(T)}{|T \setminus \bigcup \mathcal{C}|}$

Example 2: Set Cover, reducing the universe

Recall the standard greedy $O(\log n)$-approximation algorithm:

Greedy

$\mathcal{C} \leftarrow \emptyset$.
while $\mathcal{C}$ does not cover U **do**
    Find $T \in \mathcal{S}$ so as to minimize $\frac{w(T)}{|T \setminus \bigcup \mathcal{C}|}$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{T\}$.
    for each $e \in T \setminus \bigcup \mathcal{C}$ **do**
        $\mathrm{price}(e) \leftarrow \frac{w(T)}{|T \setminus \bigcup \mathcal{C}|}$

Lemma (from the standard analysis of greedy algorithm)

Let $e_1, \ldots, e_n$ be the sequence of all elements of U in the order of covering by Greedy *(ties broken arbitrarily). Then, for each $k \in 1, \ldots, n$,* $\mathrm{price}(e_k) \leq w(\mathrm{OPT})/(n - k + 1)$

Example 2: Set Cover, reducing the universe

Lemma (from the standard analysis of greedy algorithm)

Let $e_1, \ldots, e_n$ *be the sequence of all elements of* U *in the order of covering by* Greedy *(ties broken arbitrarily). Then, for each* $k \in 1, \ldots, n$*,* $\mathrm{price}(e_k) \leq w(\mathrm{OPT})/(n-k+1)$

Observation

In the early phase of Greedy elements are covered cheaply.

Example 2: Set Cover, reducing the universe

Lemma (from the standard analysis of greedy algorithm)

Let $e_1, \ldots, e_n$ *be the sequence of all elements of* U *in the order of covering by* Greedy *(ties broken arbitrarily). Then, for each* $k \in 1, \ldots, n$, $\mathrm{price}(e_k) \le w(\mathrm{OPT})/(n-k+1)$

Observation

In the early phase of Greedy elements are covered cheaply.

Exponential-Time $O(1)$-approximation

Assume we have an exact $T(n)$-time algorithm for Set Cover.

1. Run the greedy algorithm until $t \ge n/2$ elements are covered,
2. Cover the remaining elements by the exact algorithm, in time $T(n-t)$.

Example 2: Set Cover, reducing the universe

Exponential-Time $O(1)$-approximation

Assume we have an exact $T(n)$-time algorithm for Set Cover.

1. Run the greedy algorithm until $t \geq n/2$ elements are covered,
2. Cover the remaining elements by the exact algorithm, in time $T(n-t)$.

Example 2: Set Cover, reducing the universe

Exponential-Time $O(1)$-approximation

Assume we have an exact $T(n)$-time algorithm for Set Cover.

1. Run the greedy algorithm until $t \geq n/2$ elements are covered,
2. Cover the remaining elements by the exact algorithm, in time $T(n-t)$.

(Lucky) analysis

Assume we are lucky and $t = n/2$ (not bigger).

1. We pay $(H_n - H_{n/2})\mathrm{OPT} \approx (\ln n - \ln(n/2))\mathrm{OPT} = \ln 2 \cdot \mathrm{OPT}$ for the first phase,
2. we pay $\leq \mathrm{OPT}$ for the second phase.

Together we get $(1 + \ln 2)\mathrm{OPT}$.

Example 2: SET COVER, reducing the universe

Exponential-Time $O(1)$-approximation

Assume we have an exact $T(n)$-time algorithm for SET COVER.

1. Run the greedy algorithm until $t \geq n/2$ elements are covered,
2. Cover the remaining elements by the exact algorithm, in time $T(n-t)$.

Analysis

1. We pay $\leq (H_n - H_{n/2})\mathrm{OPT} \approx \ln 2 \cdot \mathrm{OPT}$ for the elements covered in phase 1, excluding the last set (that covers $e_{n/2}$),
2. We pay $\leq \mathrm{OPT}$ for the set that covers $e_{n/2}$,
3. we pay $\leq \mathrm{OPT}$ for the second phase.

Together we get $(2 + \ln 2)\mathrm{OPT}$.

Example 2: Set Cover, reducing the universe

Exponential-Time $(\ln 2 + 2)$-approximation

Assume we have an exact $T(n)$-time algorithm for Set Cover.

1. Run the greedy algorithm until $t \geq n/2$ elements are covered,
2. Cover the remaining elements by the exact algorithm, in time $T(n-t)$.

Analysis

1. We pay $\leq (H_n - H_{n/2})\mathrm{OPT} \approx \ln 2 \cdot \mathrm{OPT}$ for the elements covered in phase 1, excluding the last set (that covers $e_{n/2}$),
2. We pay $\leq \mathrm{OPT}$ for the set that covers $e_{n/2}$,
3. we pay $\leq \mathrm{OPT}$ for the second phase.

Together we get $(2 + \ln 2)\mathrm{OPT}$.

Example 2: Set Cover, reducing the universe

Exponential-Time $(\ln r + 2)$-approximation

Assume we have an exact $T(n)$-time algorithm for Set Cover.

1. Run Greedy until there are $\leq n/r$ elements not covered,
2. Cover the remaining elements by the exact algorithm, in time $T(n/r)$.

Remark 1

By stopping the Greedy algorithm when there are $\leq n/r$ uncovered elements, we get $(\ln r + 2)$-approximation in $T(n/r)$ time.

Remark 2

We show an improved algorithm with $(\ln r + 1)$-approximation in $m \times T(n/r)$ time.

Our results via instance reduction

Let $T^*(n)$ denote the time of the relevant exact algorithm, up to a polynomial factor.

1. (WEIGHTED) SET COVER:
   - r-approximation in $T^*(m/r)$ time,
   - $(1 + \ln r)$-approximation in $T^*(n/r)$ time.

Our results via instance reduction

Let $T^*(n)$ denote the time of the relevant exact algorithm, up to a polynomial factor.

1. (Weighted) Set Cover:
   - r-approximation in $T^*(m/r)$ time,
   - $(1 + \ln r)$-approximation in $T^*(n/r)$ time.
2. Bandwidth:
   - 9-approximation in $T^*(n/2)$ time.

Our results via instance reduction

Let $T^*(n)$ denote the time of the relevant exact algorithm, up to a polynomial factor.

1. (WEIGHTED) SET COVER:
   - r-approximation in $T^*(m/r)$ time,
   - $(1 + \ln r)$-approximation in $T^*(n/r)$ time.
2. BANDWIDTH:
   - 9-approximation in $T^*(n/2)$ time.
3. MAXIMUM INDEPENDENT SET:
   - r-approximation in $T^*(n/r)$-time.

Our results via instance reduction

Let $T^*(n)$ denote the time of the relevant exact algorithm, up to a polynomial factor.

1. (Weighted) Set Cover:
   - r-approximation in $T^*(m/r)$ time,
   - $(1 + \ln r)$-approximation in $T^*(n/r)$ time.
2. Bandwidth:
   - 9-approximation in $T^*(n/2)$ time.
3. Maximum Independent Set:
   - r-approximation in $T^*(n/r)$-time.
4. Vertex Coloring:
   - Björklund & Husfeldt: $(1 + \ln r)$-approximation in $\max\{T^*(n/r), O^*(2^{0.288n})\}$-time.
   - $(1 + 0.247r \ln r)$-approximation in $T^*(n/r)$-time (best for $r \in [4.05, 58)$).
   - r-approximation in $T^*(n/r)$-time (best for $r \geq 58$).

Reducing the instance: Summary

- If faster exact algorithm appears, immediately we have faster approximation.
- Approximation via instance reduction extends the applicability of (exact) exponential-time algorithms:

  Don't have enough time for running your algorithm for $n = 200$? Get approximate solution.

Reducing the instance: Open Problems

- For COLORING, in exponential time you can reduce the instance r times and get $(\ln r + 1)$-approximation (Björklund and Husfeldt). Can you do it for INDEPENDENT SET?
- Can *reduction of the instance size* be applied to BANDWIDTH? (Yes, but we have 9-approximation for reducing the graph by a half.)

Approach Two:
Cutting the Search Tree

The BANDWIDTH problem

INPUT: Graph $G = (V, E)$, integer b.
PROBLEM: Find an ordering of vertices

$$\pi : V \to \{1, \ldots, n\},$$

such that "edges have length at most b", i.e.

$$\text{for every } uv \in E,\ |\pi(u) - \pi(v)| \leq b.$$

Our results: Bandwidth

- 3/2-approximation in $O^*(5^n)$ time (poly-space),
- 2-approximation in $O^*(3^n)$ time (poly-space),
- Main result: $(4r-1)$-approximation in $O^*(2^{n/r})$ time (poly-space).

Warm-up: 2-approximation in $O^*(3^n)$ time

(Inspired the exact $O(10^n)$-time algorithm by Feige and Kilian.)

1. Divide $\{1, \ldots, n\}$ into $\lceil n/b \rceil$ intervals of length b:
   $I_j = \{jb + 1, jb + 2, \ldots, (j + 1)b\} \cap \{1, \ldots, n\}$.
2. Find an assignment of vertices to intervals such that
   - each interval I_j is assigned $|I_j|$ vertices,
   - adjacent vertices are assigned to the same interval or to neighboring intervals.

Warm-up: 2-approximation in $O^*(3^n)$ time

procedure GenerateAssignments(A)
  if for all j, $|A^{-1}(j)| = |I_j|$ **then**
    return A
  else
    $v \leftarrow$ a vertex with a neighbor w already assigned.
    if $A(w) > 0$ **then**
      GenerateAssignments($A \cup \{(v, A(w) - 1)\}$)
    GenerateAssignments($A \cup \{(v, A(w))\}$)
    if $A(w) < \lceil n/b \rceil - 1$ **then**
      GenerateAssignments($A \cup \{(v, A(w) + 1)\}$)

procedure Main
  for $j \leftarrow 0$ **to** $\lceil n/b \rceil - 1$ **do**
    GenerateAssignments ($\{(r, j)\}$)

Warm-up: 2-approximation in $O^*(3^n)$ time

1. Divide $\{1, \ldots, n\}$ into $\lceil n/b \rceil$ intervals of length b:
   $I_j = \{jb + 1, jb + 2, \ldots, (j + 1)b\} \cap \{1, \ldots, n\}$.
2. Find an assignment of vertices to intervals such that
   - Each interval I_j is assigned $|I_j|$ vertices,
   - Adjacent vertices are assigned to the same interval or to neighboring intervals.
3. Order the vertices in each interval arbitrarily.

3-approximation in $O^*(2^n)$ time

Definition

Let A be an assignment of vertices to intervals. If one can order the vertices in each interval to get an ordering π, we say π is consistent with A.

Algorithm

1. Divide $\{1, \ldots, n\}$ into $\lceil n/b \rceil$ intervals of length $2b$:
   $I_j = \{jb + 1, jb + 2, \ldots, (j + 2)b\} \cap \{1, \ldots, n\}$.
   (Note that intervals overlap.)
2. Generate a set of $O(n \cdot 2^n)$ assignments of vertices to intervals so that if the bandwith is b, then at least one of the assignments is consistent with an ordering of bandwidth b.
3. ... (to be continued) ...

3-approximation in $O^*(2^n)$ time

procedure GenerateAssignments(A)
    if all vertices are assigned **then**
        "Test(A)"
    else
        $v \leftarrow$ a vertex with a neighbor w already assigned.
        if $A(w) > 0$ **then**
            GenerateAssignments($A \cup \{(v, A(w) - 1)\}$)
        if $A(w) < \lceil n/b \rceil - 1$ **then**
            GenerateAssignments($A \cup \{(v, A(w) + 1)\}$)

procedure Main
    for $j \leftarrow 0$ **to** $\lceil n/b \rceil - 1$ **do**
        GenerateAssignments ($\{(r, j)\}$)

3-approximation in $O^*(2^n)$ time

Lemma („Testing A")

Let A be an assignment of vertices to the intervals of size $2b$.
Then there is a polynomial time algorithm such that if there is an ordering π^* of bandwidth b consistent with A, the algorithm finds an ordering π of bandwidth $3b$ consistent with A.

Proof.

1. For every edge uv, if $\max A(u) = \min A(v) - 1$, then:
   - if $|A(u)| = 2b$, replace $A(u)$ by its right half,
   - if $|A(v)| = 2b$, replace $A(v)$ by its left half.
   - (Note that π^* is still consistent with A.)
2. (now, for every edge uv, $|\max A(u) - \min A(v)| \leq 3b$)
3. Perform the standard greedy scheduling algorithm to find any ordering π consistent with A.

□

3-approximation in $O^*(2^n)$ time

Algorithm

1. Divide $\{1, \ldots, n\}$ into $\lceil n/b \rceil$ intervals of length $2b$:
   $I_j = \{jb + 1, jb + 2, \ldots, (j + 2)b\} \cap \{1, \ldots, n\}$.
   (Note that intervals overlap.)
2. Generate a set of $O(n \cdot 2^n)$ assignments of vertices to intervals so that if the bandwith is b, then at least one of the assignments is consistent with an ordering of bandwidth b.
3. Apply the lemma to each of the assignments.

Approximation scheme

Theorem

For any $r \in \mathbb{N}$, there is a $(4r-1)$-approximation algorithm in $O^*(2^{n/r})$ time.

(Details skipped here)

The end

Thank you for your attention!