Approximation Algorithms (ADM III)
1- Introduction

Guillaume Sagnol

Technische Universität Berlin

Outline

Organization

- Short Videos on ISIS

Organization

- Short Videos on ISIS
- Weekly Zoom-Slot
  - Wednesday 12:00 (noon) ?
  - FAQ-sessions
  - Addional material
  - Exercises

Organization

- Short Videos on ISIS
- Weekly Zoom-Slot
  - Wednesday 12:00 (noon) ?
  - FAQ-sessions
  - Addional material
  - Exercises
- Set of problems as homework once in a while

Organization

- Short Videos on ISIS
- Weekly Zoom-Slot
  - Wednesday 12:00 (noon) ?
  - FAQ-sessions
  - Addional material
  - Exercises
- Set of problems as homework once in a while
- Final oral exam in spring during the semester break (details t.b.a.)

Literature

The course is based on selected parts of the textbook:

- David P. Williamson, David B. Shmoys, The Design of Approximation Algorithms, Cambridge University Press, 2011

Further reading:

- V. V. Vazirani, Approximation Algorithms, Springer Verlag, 2001
- G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties, Springer Verlag, 1999
- D. S. Hochbaum (ed.), Approximation Algorithms for NP-Hard Problems, PWS Publishing Company, 1995
- B. Korte, J. Vygen, Combinatorial Optimization, Springer Verlag, fifth edition, 2012

Outline

How to Tackle *NP*-Hard Optimization Problems?

Most interesting discrete optimization problems are *NP*-hard.

How to Tackle *NP*-Hard Optimization Problems?

Most interesting discrete optimization problems are *NP*-hard.

Thus, unless $P = NP$, we cannot find algorithms that simultaneously

1. find optimal solutions,
2. in polynomial time,
3. for any instance.

How to Tackle *NP*-Hard Optimization Problems?

Most interesting discrete optimization problems are *NP*-hard.

Thus, unless $P = NP$, we cannot find algorithms that simultaneously

1. find optimal solutions,
2. in polynomial time,
3. for any instance.

Therefore, we need to relax at least one of the three requirements.

How to Tackle *NP*-Hard Optimization Problems?

Most interesting discrete optimization problems are *NP*-hard.

Thus, unless $P = NP$, we cannot find algorithms that simultaneously

1. find optimal solutions,
2. in polynomial time,
3. for any instance.

Therefore, we need to relax at least one of the three requirements.

We study approximation algorithms, i.e., we relax the first requirement and search for algorithms that produce solutions that are "good enough".

How to Tackle *NP*-Hard Optimization Problems?

Most interesting discrete optimization problems are *NP*-hard.

Thus, unless $P = NP$, we cannot find algorithms that simultaneously

1. find optimal solutions,
2. in polynomial time,
3. for any instance.

Therefore, we need to relax at least one of the three requirements.

We study approximation algorithms, i.e., we relax the first requirement and search for algorithms that produce solutions that are "good enough".

What is "good enough"?
We would like to have some sort of a priori performance guarantee.

Approximation Algorithms

Definition 1.1

An α-approximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the optimum value.

Approximation Algorithms

Definition 1.1

An α-approximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the optimum value.

For an α-approximation algorithm, we call α the performance guarantee (or approximation ratio/factor) of the algorithm.

Approximation Algorithms

Definition 1.1

An α-approximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the optimum value.

For an α-approximation algorithm, we call α the performance guarantee (or approximation ratio/factor) of the algorithm.

Convention:

- $\alpha > 1$ for minimization problems, and
- $\alpha < 1$ for maximization problems.

Approximation Algorithms

Definition 1.1

An α-approximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor of α of the optimum value.

For an α-approximation algorithm, we call α the performance guarantee (or approximation ratio/factor) of the algorithm.

Convention:

- $\alpha > 1$ for minimization problems, and
- $\alpha < 1$ for maximization problems.

Thus, a $\frac{1}{2}$-approximation algorithm for a maximization problem is a polynomial-time algorithm that always returns a solution whose value is at least half the value of the optimum value.

Approximation Algorithms

Given an optimization problem, we'll often use the notation $\texttt{ALG}$ to denote the value returned by a given algorithm for this problem, and $\texttt{OPT}$ to denote the optimum value of the problem.

Example 1.2

For a maximization problem, an α-approximation algorithm satisfies

$$\texttt{ALG} \geq \alpha\,\texttt{OPT}$$

for all instances of the problem.

Approximation Algorithms

Given an optimization problem, we'll often use the notation $\mathtt{ALG}$ to denote the value returned by a given algorithm for this problem, and $\mathtt{OPT}$ to denote the optimum value of the problem.

Example 1.2

For a maximization problem, an α-approximation algorithm satisfies

$$\mathtt{ALG} \geq \alpha\,\mathtt{OPT}$$

for all instances of the problem.

Example:

Constructing an inclusion-wise maximal matching is a $\frac{1}{2}$-approx. algorithm for the maximum cardinality matching problem.

Approximation factors

Remarks:

- The worst-case bounds are often due to pathological cases that hardly arise in practice.

Approximation factors

Remarks:

- The worst-case bounds are often due to pathological cases that hardly arise in practice.
- Often, approximation algorithms are much better in practice than indicated by their performance guarantee.

Approximation factors

Remarks:

- The worst-case bounds are often due to pathological cases that hardly arise in practice.
- Often, approximation algorithms are much better in practice than indicated by their performance guarantee.
- Nevertheless, the study of worst-case bounds provides a mathematically rigourous basis to analyse heuristics, and to undersand *what can go wrong*.

Approximation factors

Remarks:

- The worst-case bounds are often due to pathological cases that hardly arise in practice.
- Often, approximation algorithms are much better in practice than indicated by their performance guarantee.
- Nevertheless, the study of worst-case bounds provides a mathematically rigourous basis to analyse heuristics, and to undersand *what can go wrong*.
- Approximation factors and inapproximability results give a kind of *metric* to compare the hardness of discrete optimization problems.

Overview of lecture

Learn *techniques* to design approximation algorithms

- Greedy algorithms
- Dynamic Programming & Data rounding
- Linear Programming based techniques (dual fitting, primal-dual algos)
- Deterministic & Randomized Rounding
- Reductions to show *hardness of approximation*

Overview of lecture

Learn *techniques* to design approximation algorithms

- Greedy algorithms
- Dynamic Programming & Data rounding
- Linear Programming based techniques (dual fitting, primal-dual algos)
- Deterministic & Randomized Rounding
- Reductions to show *hardness of approximation*

Approximation results for many standard discrete problems, e.g.

- Set cover
- Scheduling
- Facility location
- Traveling Salesperson Problem
- Clustering
- ...

Polynomial-Time Approximation Schemes

Definition 1.3 (PTAS)

A polynomial-time approximation scheme (PTAS) is a family of algorithms $\{A_\varepsilon\}$, where there is an algorithm for each $\varepsilon > 0$, such that A_ε is a

- $(1+\varepsilon)$-approximation algorithm (for minimization problems), or a
- $(1-\varepsilon)$-approximation algorithm (for maximization problems).

Polynomial-Time Approximation Schemes

Definition 1.3 (PTAS)

A polynomial-time approximation scheme (PTAS) is a family of algorithms $\{A_\varepsilon\}$, where there is an algorithm for each $\varepsilon > 0$, such that A_ε is a

- $(1 + \varepsilon)$-approximation algorithm (for minimization problems), or a
- $(1 - \varepsilon)$-approximation algorithm (for maximization problems).

Examples: PTASes exist for the Euclidean TSP and the planar MAXIMUM INDEPENDENT SET.

Polynomial-Time Approximation Schemes

Definition 1.3 (PTAS)

A polynomial-time approximation scheme (PTAS) is a family of algorithms $\{A_\varepsilon\}$, where there is an algorithm for each $\varepsilon > 0$, such that A_ε is a

- $(1+\varepsilon)$-approximation algorithm (for minimization problems), or a
- $(1-\varepsilon)$-approximation algorithm (for maximization problems).

Examples: PTASes exist for the Euclidean TSP and the planar MAXIMUM INDEPENDENT SET.

Remark: The running time of A_ε may depend badly on ε, e.g. $\{A_\varepsilon\}$ is a PTAS if A_ε runs in $O(n^{1/\varepsilon})$ or even $O(n^{\exp(1/\varepsilon)})$.

Polynomial-Time Approximation Schemes

Definition 1.4 (FPTAS)

A fully polynomial-time approximation scheme (FPTAS) is a family of algorithms $\{A_\varepsilon\}$, where there is an algorithm for each $\varepsilon > 0$, such that the running time of A_ε is polynomial in both the problem size n and $\frac{1}{\varepsilon}$, and A_ε is a $(1+\varepsilon)$-approximation algorithm for minimization problems, (resp. a $(1-\varepsilon)$-approximation algorithm for maximization problems).

Polynomial-Time Approximation Schemes

Definition 1.4 (FPTAS)

A fully polynomial-time approximation scheme (FPTAS) is a family of algorithms $\{A_\varepsilon\}$, where there is an algorithm for each $\varepsilon > 0$, such that the running time of A_ε is polynomial in both the problem size n and $\frac{1}{\varepsilon}$, and A_ε is a $(1+\varepsilon)$-approximation algorithm for minimization problems, (resp. a $(1-\varepsilon)$-approximation algorithm for maximization problems).

Examples: FPTASes exist for the Knapsack Problem and for scheduling on a fixed number of identical machines.

Polynomial-Time Approximation Schemes

Definition 1.4 (FPTAS)

A fully polynomial-time approximation scheme (FPTAS) is a family of algorithms $\{A_\varepsilon\}$, where there is an algorithm for each $\varepsilon > 0$, such that the running time of A_ε is polynomial in both the problem size n and $\frac{1}{\varepsilon}$, and A_ε is a $(1+\varepsilon)$-approximation algorithm for minimization problems, (resp. a $(1-\varepsilon)$-approximation algorithm for maximization problems).

Examples: FPTASes exist for the Knapsack Problem and for scheduling on a fixed number of identical machines.

Remarks: The running time of an FPTAS is of the form $O(1/\varepsilon^k \cdot n^d)$, so in some sense, the precision ε only influences the hidden constant of the polynomial.

Outline

Set Cover Problem

- There are several fundamental techniques used in the design and analysis of approximation algorithms.

Set Cover Problem

- There are several fundamental techniques used in the design and analysis of approximation algorithms.
- In particular, linear programming plays an essential role!

Set Cover Problem

- There are several fundamental techniques used in the design and analysis of approximation algorithms.
- In particular, linear programming plays an essential role!
- In this introductory chapter, we illustrate some of the techniques on the Set Cover Problem.

Set Cover Problem

- There are several fundamental techniques used in the design and analysis of approximation algorithms.
- In particular, linear programming plays an essential role!
- In this introductory chapter, we illustrate some of the techniques on the Set Cover Problem.

Set Cover Problem:
Given: A set of elements $E = \{e_1, \dots, e_n\}$, a family of subsets $\{S_1, \dots, S_m\} \subseteq 2^E$, and a weight $w_j \geq 0$ for each $j \in \{1, \dots, m\}$.

Set Cover Problem

- There are several fundamental techniques used in the design and analysis of approximation algorithms.
- In particular, linear programming plays an essential role!
- In this introductory chapter, we illustrate some of the techniques on the Set Cover Problem.

Set Cover Problem:
Given: A set of elements $E = \{e_1, \ldots, e_n\}$, a family of subsets $\{S_1, \ldots, S_m\} \subseteq 2^E$, and a weight $w_j \geq 0$ for each $j \in \{1, \ldots, m\}$.

Task: Find $I \subseteq \{1, \ldots, m\}$ minimizing $\sum_{j \in I} w_j$ s.t. $\bigcup_{j \in I} S_j = E$.

Set Cover Problem

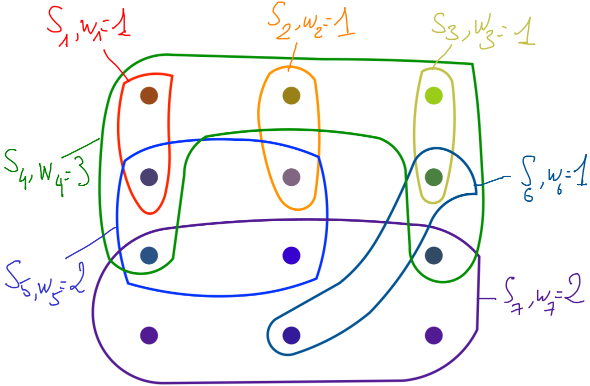
- There are several fundamental techniques used in the design and analysis of approximation algorithms.
- In particular, linear programming plays an essential role!
- In this introductory chapter, we illustrate some of the techniques on the Set Cover Problem.

Set Cover Problem:
Given: A set of elements $E = \{e_1, \ldots, e_n\}$, a family of subsets $\{S_1, \ldots, S_m\} \subseteq 2^E$, and a weight $w_j \geq 0$ for each $j \in \{1, \ldots, m\}$.

Task: Find $I \subseteq \{1, \ldots, m\}$ minimizing $\sum_{j \in I} w_j$ s.t. $\bigcup_{j \in I} S_j = E$.

If $w_j = 1$ for each $j \in \{1, \ldots, m\}$, the problem is called Unweighted Set Cover Problem.

Set Cover



Cost of two set covers:

- S_1, S_2, S_3, S_7: $W = 5$ (minimal cost for this example)
- S_2, S_4, S_7: $W = 6$

Outline

IP Formulation of Set Cover

Formulation as integer linear program:

$$
\begin{aligned}
z_{IP}^* = \quad \min \quad & \sum_{j=1}^{m} w_j \cdot x_j \\
\text{s.t.} \quad & \sum_{j:e_i \in S_j} x_j \geq 1 \quad \text{for all } i = 1, \ldots, n \\
& x_j \in \{0, 1\} \quad \text{for all } j = 1, \ldots, m
\end{aligned} \tag{1}
$$

IP Formulation of Set Cover

Formulation as integer linear program:

$$z_{IP}^* = \min \sum_{j=1}^{m} w_j \cdot x_j$$

$$\text{s.t.} \sum_{j: e_i \in S_j} x_j \geq 1 \quad \text{for all } i = 1, \ldots, n \tag{1}$$

$$x_j \in \{0, 1\} \quad \text{for all } j = 1, \ldots, m$$

Linear programming relaxation:

$$z_{LP}^* = \min \sum_{j=1}^{m} w_j \cdot x_j$$

$$\text{s.t.} \sum_{j: e_i \in S_j} x_j \geq 1 \quad \text{for all } i = 1, \ldots, n \tag{2}$$

$$x_j \geq 0 \quad \text{for all } j = 1, \ldots, m$$

IP Formulation of Set Cover

Formulation as integer linear program:

$$
\begin{aligned}
z_{IP}^* = \quad \min \quad & \sum_{j=1}^{m} w_j \cdot x_j \\
\text{s.t.} \quad & \sum_{j: e_i \in S_j} x_j \geq 1 \qquad \text{for all } i = 1, \ldots, n \\
& x_j \in \{0, 1\} \qquad \text{for all } j = 1, \ldots, m
\end{aligned} \tag{1}
$$

Linear programming relaxation:

$$
\begin{aligned}
z_{LP}^* = \quad \min \quad & \sum_{j=1}^{m} w_j \cdot x_j \\
\text{s.t.} \quad & \sum_{j: e_i \in S_j} x_j \geq 1 \qquad \text{for all } i = 1, \ldots, n \\
& x_j \geq 0 \qquad \text{for all } j = 1, \ldots, m
\end{aligned} \tag{2}
$$

A Deterministic Rounding Algorithm

For each $i = 1, \ldots, n$ let $f_i := |\{j \mid e_i \in S_j\}|$ be the number of sets containing e_i,

A Deterministic Rounding Algorithm

For each $i = 1, \ldots, n$ let $f_i := |\{j \mid e_i \in S_j\}|$ be the number of sets containing e_i, and $f := \max_{i=1,\ldots,n} f_i$.

A Deterministic Rounding Algorithm

For each $i = 1, \ldots, n$ let $f_i := |\{j \mid e_i \in S_j\}|$ be the number of sets containing e_i, and $f := \max_{i=1,\ldots,n} f_i$.

Deterministic Rounding Algorithm for Set Cover:

1. Compute an optimal solution x^* to the set-cover-LP (2).

A Deterministic Rounding Algorithm

For each $i = 1, \ldots, n$ let $f_i := |\{j \mid e_i \in S_j\}|$ be the number of sets containing e_i, and $f := \max_{i=1,\ldots,n} f_i$.

Deterministic Rounding Algorithm for Set Cover:

1. Compute an optimal solution x^* to the set-cover-LP (2).
2. For each $j \in \{1, \ldots, m\}$, set $\hat{x}_j = 1$ if $x_j^* \geq \dfrac{1}{f}$, and $\hat{x}_j = 0$ otherwise.

A Deterministic Rounding Algorithm

For each $i = 1, \ldots, n$ let $f_i := |\{j \mid e_i \in S_j\}|$ be the number of sets containing e_i, and $f := \max_{i=1,\ldots,n} f_i$.

Deterministic Rounding Algorithm for Set Cover:

1. Compute an optimal solution x^* to the set-cover-LP (2).
2. For each $j \in \{1, \ldots, m\}$, set $\hat{x}_j = 1$ if $x_j^* \geq \frac{1}{f}$, and $\hat{x}_j = 0$ otherwise.

Lemma 1.5

The collection of subsets S_j with $j \in \hat{I} := \{j \mid \hat{x}_j = 1\}$ is a set cover.

A Deterministic Rounding Algorithm

For each $i = 1, \ldots, n$ let $f_i := |\{j \mid e_i \in S_j\}|$ be the number of sets containing e_i, and $f := \max_{i=1,\ldots,n} f_i$.

Deterministic Rounding Algorithm for Set Cover:

1. Compute an optimal solution x^* to the set-cover-LP (2).
2. For each $j \in \{1, \ldots, m\}$, set $\hat{x}_j = 1$ if $x_j^* \geq \frac{1}{f}$, and $\hat{x}_j = 0$ otherwise.

Lemma 1.5

The collection of subsets S_j with $j \in \hat{I} := \{j \mid \hat{x}_j = 1\}$ is a set cover.

Theorem 1.6

The rounding algorithm above is an f-approximation algorithm for the Set Cover Problem.

Proof...

Vertex cover

Input: Graph $G = (V, E)$, weight $w_i \geq 0$ for all $i \in V$.

Task: Find $I \subset V$ minimizing $w(I) := \sum_{i \in I} w_i$,

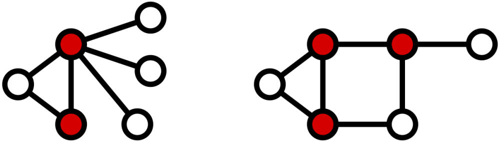
such that every edge $e \in E$ has at least one end point in I.

Vertex cover

Input: Graph $G = (V, E)$, weight $w_i \geq 0$ for all $i \in V$.

Task: Find $I \subset V$ minimizing $w(I) := \sum_{i \in I} w_i$,

such that every edge $e \in E$ has at least one end point in I.



Some *vertex covers* (minimal for unweighted problem, i.e., $w_i = 1$)

Exercise 1.7

Use the previous algorithm to design an approximation algorithm for VERTEX COVER. What is its performance guarantee ?

Outline

Dual Linear Program

Linear programming relaxation:

$$
\begin{aligned}
z_{LP}^* = \quad \min \quad & \sum_{j=1}^{m} w_j \cdot x_j \\
\text{s.t.} \quad & \sum_{j:e_i \in S_j} x_j \geq 1 \qquad \text{for all } i = 1, \ldots, n \\
& x_j \geq 0 \qquad \text{for all } j = 1, \ldots, m
\end{aligned}
$$

Dual Linear Program

Linear programming relaxation:

$$\begin{aligned} z^*_{LP} = \quad \min \quad & \sum_{j=1}^{m} w_j \cdot x_j \\ \text{s.t.} \quad & \sum_{j:e_i \in S_j} x_j \geq 1 \qquad \text{for all } i = 1, \ldots, n \\ & x_j \geq 0 \qquad \text{for all } j = 1, \ldots, m \end{aligned}$$

Dual linear program:

$$\begin{aligned} z^*_{LP} = \quad \max \quad & \sum_{i=1}^{n} y_i \\ \text{s.t.} \quad & \sum_{e_i \in S_j} y_i \leq w_j \qquad \text{for all } j = 1, \ldots, m \\ & y_i \geq 0 \qquad \text{for all } i = 1, \ldots, n \end{aligned} \tag{3}$$

Rounding a Dual Solution

Dual Rounding Algorithm for Set Cover:

1. compute an optimal solution y^* to the dual (3) of the set-cover-LP;

Rounding a Dual Solution

Dual Rounding Algorithm for Set Cover:

1. compute an optimal solution y^* to the dual (3) of the set-cover-LP;
2. let $I^* := \{j \mid \sum_{i:e_i \in S_j} y_i^* = w_j\}$;

Rounding a Dual Solution

Dual Rounding Algorithm for Set Cover:

1. compute an optimal solution y^* to the dual (3) of the set-cover-LP;
2. let $I^* := \{j \mid \sum_{i:e_i \in S_j} y_i^* = w_j\}$;

Lemma 1.8

The collection of subsets S_j with $j \in I^*$ is a set cover.

Rounding a Dual Solution

Dual Rounding Algorithm for Set Cover:

1. compute an optimal solution y^* to the dual (3) of the set-cover-LP;
2. let $I^* := \{j \mid \sum_{i : e_i \in S_j} y_i^* = w_j\}$;

Lemma 1.8

The collection of subsets S_j with $j \in I^*$ is a set cover.

Theorem 1.9

The dual rounding algorithm is an f-approximation algorithm for the Set Cover Problem.

Rounding a Dual Solution

Dual Rounding Algorithm for Set Cover:

1. compute an optimal solution y^* to the dual (3) of the set-cover-LP;
2. let $I^* := \{j \mid \sum_{i : e_i \in S_j} y_i^* = w_j\}$;

Lemma 1.8

The collection of subsets S_j with $j \in I^*$ is a set cover.

Theorem 1.9

The dual rounding algorithm is an f-approximation algorithm for the Set Cover Problem.

Remark: Notice that $I^* \supseteq \hat{I}$ (the solution obtained by rounding the primal LP) due to complementary slackness!

Proof...

Outline

Primal-Dual Algorithm

Note: The two previous algorithms require solving a linear program.

Primal-Dual Algorithm

Note: The two previous algorithms require solving a linear program. Special purpose algorithms are often much faster!

Primal-Dual Algorithm

Note: The two previous algorithms require solving a linear program. Special purpose algorithms are often much faster!

Idea: Construct a feasible dual solution that is “good enough”.

Primal-Dual Algorithm

Note: The two previous algorithms require solving a linear program. Special purpose algorithms are often much faster!

Idea: Construct a feasible dual solution that is "good enough".

Primal-dual algorithm for the Set Cover Problem:

1. set $y :\equiv 0$ and $I := \emptyset$;
2. while $\exists e_k \notin \bigcup_{j \in I} S_j$
3. increase y_k until $\exists j$ with $e_k \in S_j$ such that $\sum_{i : e_i \in S_j} y_i = w_j$;
4. set $I := I \cup \{j\}$;

Primal-Dual Algorithm

Note: The two previous algorithms require solving a linear program. Special purpose algorithms are often much faster!

Idea: Construct a feasible dual solution that is "good enough".

Primal-dual algorithm for the Set Cover Problem:

1. set $y :\equiv 0$ and $I := \emptyset$;
2. while $\exists e_k \notin \bigcup_{j \in I} S_j$
3. increase y_k until $\exists j$ with $e_k \in S_j$ such that $\sum_{i : e_i \in S_j} y_i = w_j$;
4. set $I := I \cup \{j\}$;

Theorem 1.10

The primal-dual algorithm is an f-approximation algorithm for the Set Cover Problem.

Outline

Greedy Algorithm

Idea: Iteratively select a set minimizing the ratio of its weight to the number of currently uncovered elements it contains.

Greedy Algorithm

Idea: Iteratively select a set minimizing the ratio of its weight to the number of currently uncovered elements it contains.

Greedy algorithm for the Set Cover Problem

1 set $I := \emptyset$ and $\hat{S}_j := S_j$ for all j;

Greedy Algorithm

Idea: Iteratively select a set minimizing the ratio of its weight to the number of currently uncovered elements it contains.

Greedy algorithm for the Set Cover Problem

1. set $I := \emptyset$ and $\hat{S}_j := S_j$ for all j;
2. while I is not a cover

Greedy Algorithm

Idea: Iteratively select a set minimizing the ratio of its weight to the number of currently uncovered elements it contains.

Greedy algorithm for the Set Cover Problem

1. set $I := \emptyset$ and $\hat{S}_j := S_j$ for all j;
2. while I is not a cover
3. $\quad \ell := \operatorname{argmin}\left\{ \frac{w_j}{|\hat{S}_j|} \,\middle|\, \hat{S}_j \neq \emptyset \right\}$;
4. $\quad$ set $I := I \cup \{\ell\}$;
5. $\quad$ set $\hat{S}_j := \hat{S}_j \setminus S_\ell$ for all j;

Greedy Algorithm

Idea: Iteratively select a set minimizing the ratio of its weight to the number of currently uncovered elements it contains.

Greedy algorithm for the Set Cover Problem

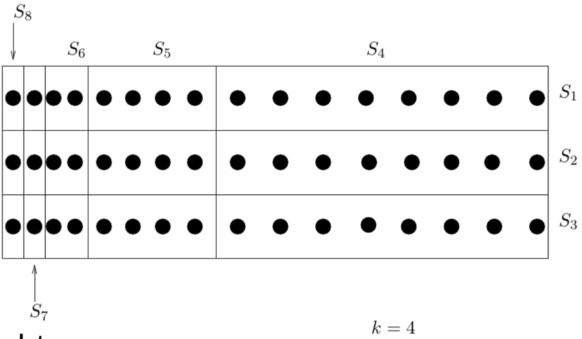
1. set $I := \emptyset$ and $\hat{S}_j := S_j$ for all j;
2. while I is not a cover
3. $\quad \ell := \operatorname{argmin}\left\{ \frac{w_j}{|\hat{S}_j|} \,\middle|\, \hat{S}_j \neq \emptyset \right\}$;
4. $\quad$ set $I := I \cup \{\ell\}$;
5. $\quad$ set $\hat{S}_j := \hat{S}_j \setminus S_\ell$ for all j;

Theorem 1.11

The greedy algorithm returns a cover I with $w(I) \leq H_g \cdot z^*_{LP}$, where

$$g := \max_j |S_j| \text{ and } H_g := \sum_{k=1}^{g} \frac{1}{k} \approx \ln\ g.$$

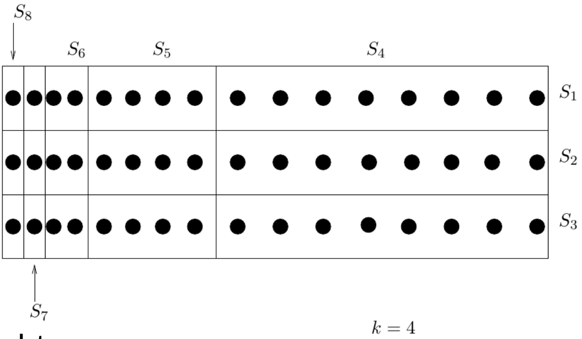
Tightness of approximation factor



Instance:

- unit weights
- $E = \{(i,j) : i \in [3],\ j \in [2^k]\}$
- $S_1 = \{(1,j) j \in [2^k]\}, S_2 = \{(2,j) j \in [2^k]\}, S_3 = \{(3,j) j \in [2^k]\}$
- S_{3+j} contains the points of 2^{k-j} "columns", $\forall j \in [k]$
- S_{k+4} contains the points of the last remaining column.

Tightness of approximation factor



Instance:

- unit weights
- $E = \{(i,j) : i \in [3],\ j \in [2^k]\}$
- $S_1 = \{(1,j) j \in [2^k]\}, S_2 = \{(2,j) j \in [2^k]\}, S_3 = \{(3,j) j \in [2^k]\}$
- S_{3+j} contains the points of 2^{k-j} "columns", $\forall j \in [k]$
- S_{k+4} contains the points of the last remaining column.

GREEDY selects $S_4, \ldots, S_{4+k}$ but the optimal set cover is (S_1, S_2, S_3), hence an approx. ratio of $(k+1)/3 = O(\log(n))$.

Proof...

Outline

Randomized Rounding Algorithm

Idea for an algorithm:

1. Compute an optimal solution x^* to the set-cover-LP (2).

Randomized Rounding Algorithm

Idea for an algorithm:

1. Compute an optimal solution x^* to the set-cover-LP (2).
2. Take S_j into the set cover solution with probability x_j^*.

Randomized Rounding Algorithm

Idea for an algorithm:

1. Compute an optimal solution x^* to the set-cover-LP (2).
2. Take S_j into the set cover solution with probability x_j^*.

Lemma 1.12

The expected value of the computed solution is equal to z_{LP}^*. Any element e_i is covered with probability at least $1 - e^{-1}$.

Randomized Rounding Algorithm

Idea for an algorithm:

1. Compute an optimal solution x^* to the set-cover-LP (2).
2. Take S_j into the set cover solution with probability x_j^*.

Lemma 1.12

The expected value of the computed solution is equal to z_{LP}^*. Any element e_i is covered with probability at least $1 - e^{-1}$.

Refined algorithm:

For some constant $c \geq 2$, take S_j with probability $1 - (1 - x_j^*)^{c \ln n}$.

Randomized Rounding Algorithm

Idea for an algorithm:

1. Compute an optimal solution x^* to the set-cover-LP (2).
2. Take S_j into the set cover solution with probability x_j^*.

Lemma 1.12

The expected value of the computed solution is equal to z_{LP}^*. Any element e_i is covered with probability at least $1 - e^{-1}$.

Refined algorithm:

For some constant $c \geq 2$, take S_j with probability $1 - (1 - x_j^*)^{c \ln n}$.

Theorem 1.13

The refined algorithm is a randomized $O(\ln n)$-approximation algorithm that produces a set cover with high probability.

Randomized Rounding Algorithm

Idea for an algorithm:

1. Compute an optimal solution x^* to the set-cover-LP (2).
2. Take S_j into the set cover solution with probability x_j^*.

Lemma 1.12

The expected value of the computed solution is equal to z_{LP}^*. Any element e_i is covered with probability at least $1 - e^{-1}$.

Refined algorithm:

For some constant $c \geq 2$, take S_j with probability $1 - (1 - x_j^*)^{c \ln n}$.

Theorem 1.13

The refined algorithm is a randomized $O(\ln n)$-approximation algorithm that produces a set cover with high probability.

Proof...

Outline

Hardness of Approximability Results for Set Cover

Theorem 1.14(Lund & Yannakakis 1994)

If there is a $(c \ln n)$-approximation algorithm for the Unweighted Set Cover Problem for some constant $c < 1$, then there is an $O(n^{O(\log \log n)})$-time deterministic algorithm for each NP-complete problem.

Hardness of Approximability Results for Set Cover

Theorem 1.14(Lund & Yannakakis 1994)

If there is a $(c \ln n)$-approximation algorithm for the Unweighted Set Cover Problem for some constant $c < 1$, then there is an $O(n^{O(\log \log n)})$-time deterministic algorithm for each NP-complete problem.

Theorem 1.15 (Feige 1998)

There is some constant $c > 0$ such that if there is a $(c \ln n)$-approximation algorithm for the Unweighted Set Cover Problem, then $P = NP$.

Hardness of Approximability Results for Vertex Cover

Theorem 1.16 (Dinur & Safra 2002)

If there is an α-approximation algorithm for the Vertex Cover Problem with $\alpha < 10\sqrt{5} - 21 \approx 1.36$, then $P = NP$.

Hardness of Approximability Results for Vertex Cover

Theorem 1.16 (Dinur & Safra 2002)

If there is an α-approximation algorithm for the Vertex Cover Problem with $\alpha < 10\sqrt{5} - 21 \approx 1.36$, then $P = NP$.

Theorem 1.17 (Khot & Regev 2008)

Assuming the Unique Games Conjecture holds, if there is an α-approximation algorithm for the Vertex Cover Problem with $\alpha < 2$, then $P = NP$.