Approximation Algorithms (ADM III)

4- Deterministic Rounding of Linear Programs

Guillaume Sagnol

Outline

The scheduling problem $1|r_j|\sum_j C_j$

Given: jobs with processing time $p_j > 0$, release date $r_j \geq 0$, $j = 1, \ldots, n$.

Task: Schedule the jobs nonpreemptively on a single machine;

minimize the *total completion time* $\sum_{j=1}^{n} C_j$.

The scheduling problem $1|r_j|\sum_j C_j$

Given: jobs with processing time $p_j > 0$, release date $r_j \geq 0$, $j = 1, \ldots, n$.

Task: Schedule the jobs nonpreemptively on a single machine;

minimize the *total completion time* $\sum_{j=1}^{n} C_j$.

Remarks:

- This problem is known to be strongly NP-hard.

The scheduling problem $1|r_j|\sum_j C_j$

Given: jobs with processing time $p_j > 0$, release date $r_j \geq 0$, $j = 1, \ldots, n$.

Task: Schedule the jobs nonpreemptively on a single machine;

minimize the *total completion time* $\displaystyle\sum_{j=1}^{n} C_j$.

Remarks:

- This problem is known to be strongly NP-hard.
- The preemptive relaxation, however, can be solved efficiently.

The scheduling problem $1|r_j|\sum_j C_j$

Given: jobs with processing time $p_j > 0$, release date $r_j \geq 0$, $j = 1, \ldots, n$.

Task: Schedule the jobs nonpreemptively on a single machine;

minimize the *total completion time* $\sum_{j=1}^{n} C_j$.

Remarks:

- This problem is known to be strongly NP-hard.
- The preemptive relaxation, however, can be solved efficiently.

Shortest Remaining Processing Time (SRPT) Rule

- At any point in time, process an available and uncompleted job with shortest remaining processing time.

The scheduling problem $1|r_j|\sum_j C_j$

Given: jobs with processing time $p_j > 0$, release date $r_j \geq 0$, $j = 1, \ldots, n$.

Task: Schedule the jobs nonpreemptively on a single machine;

minimize the *total completion time* $\sum_{j=1}^{n} C_j$.

Remarks:

- This problem is known to be strongly NP-hard.
- The preemptive relaxation, however, can be solved efficiently.

Shortest Remaining Processing Time (SRPT) Rule

- At any point in time, process an available and uncompleted job with shortest remaining processing time.

Theorem 4.1

The SRPT Rule finds an optimal preemptive schedule in time $O(n \log n)$.

The scheduling problem $1|r_j|\sum_j C_j$

Given: jobs with processing time $p_j > 0$, release date $r_j \geq 0$, $j = 1, \ldots, n$.

Task: Schedule the jobs nonpreemptively on a single machine;

minimize the *total completion time* $\sum_{j=1}^{n} C_j$.

Remarks:

- This problem is known to be strongly NP-hard.
- The preemptive relaxation, however, can be solved efficiently.

Shortest Remaining Processing Time (SRPT) Rule

- At any point in time, process an available and uncompleted job with shortest remaining processing time.

Theorem 4.1

The SRPT Rule finds an optimal preemptive schedule in time $O(n \log n)$.

Proof: Use an exchange argument. □

Converting Preemptive into Nonpreemptive Schedule

Idea: Use optimal preemptive solution to get good nonpreemptive solution.

Converting Preemptive into Nonpreemptive Schedule

Idea: Use optimal preemptive solution to get good nonpreemptive solution.

Algorithm

1. compute optimal preemptive schedule with job completion times C_j^P;

Converting Preemptive into Nonpreemptive Schedule

Idea: Use optimal preemptive solution to get good nonpreemptive solution.

Algorithm

1. compute optimal preemptive schedule with job completion times C_j^P;
2. sort jobs such that $C_1^P < C_2^P < \cdots < C_n^P$;

Converting Preemptive into Nonpreemptive Schedule

Idea: Use optimal preemptive solution to get good nonpreemptive solution.

Algorithm

1. compute optimal preemptive schedule with job completion times C_j^P;
2. sort jobs such that $C_1^P < C_2^P < \cdots < C_n^P$;
3. schedule all jobs nonpreemptively and as early as possible in this order;

Converting Preemptive into Nonpreemptive Schedule

Idea: Use optimal preemptive solution to get good nonpreemptive solution.

Algorithm

1. compute optimal preemptive schedule with job completion times C_j^P;
2. sort jobs such that $C_1^P < C_2^P < \cdots < C_n^P$;
3. schedule all jobs nonpreemptively and as early as possible in this order;

Step 3: set $C_1 := r_1 + p_1$; for $j = 2$ to n set $C_j := \max\{r_j, C_{j-1}\} + p_j$;

Converting Preemptive into Nonpreemptive Schedule

Idea: Use optimal preemptive solution to get good nonpreemptive solution.

Algorithm

1. compute optimal preemptive schedule with job completion times C_j^P;
2. sort jobs such that $C_1^P < C_2^P < \cdots < C_n^P$;
3. schedule all jobs nonpreemptively and as early as possible in this order;

Step 3: set $C_1 := r_1 + p_1$; for $j = 2$ to n set $C_j := \max\{r_j, C_{j-1}\} + p_j$;

Lemma 4.2

For each job $j = 1, \ldots, n$, it holds that $C_j \leq 2 \cdot C_j^P$.

Converting Preemptive into Nonpreemptive Schedule

Idea: Use optimal preemptive solution to get good nonpreemptive solution.

Algorithm

1. compute optimal preemptive schedule with job completion times C_j^P;
2. sort jobs such that $C_1^P < C_2^P < \cdots < C_n^P$;
3. schedule all jobs nonpreemptively and as early as possible in this order;

Step 3: set $C_1 := r_1 + p_1$; for $j = 2$ to n set $C_j := \max\{r_j, C_{j-1}\} + p_j$;

Lemma 4.2

For each job $j = 1, \dots, n$, it holds that $C_j \leq 2 \cdot C_j^P$.

Theorem 4.3

The algorithm above is a 2-approximation algorithm.

Converting Preemptive into Nonpreemptive Schedule

Idea: Use optimal preemptive solution to get good nonpreemptive solution.

Algorithm

1. compute optimal preemptive schedule with job completion times C_j^P;
2. sort jobs such that $C_1^P < C_2^P < \cdots < C_n^P$;
3. schedule all jobs nonpreemptively and as early as possible in this order;

Step 3: set $C_1 := r_1 + p_1$; for $j = 2$ to n set $C_j := \max\{r_j, C_{j-1}\} + p_j$;

Lemma 4.2

For each job $j = 1, \ldots, n$, it holds that $C_j \leq 2 \cdot C_j^P$.

Theorem 4.3

The algorithm above is a 2-approximation algorithm.

Proof:… □

Outline

Problem $1|r_j|\sum_j w_j C_j$

Given: As before, but now all jobs j also have a weight $w_j \geq 0$.

Problem $1|r_j|\sum_j w_j C_j$

Given: As before, but now all jobs j also have a weight $w_j \geq 0$.

Task: Minimize the total *weighted* completion time $\sum_{j=1}^{n} w_j C_j$.

Problem $1|r_j|\sum_j w_j C_j$

Given: As before, but now all jobs j also have a weight $w_j \geq 0$.

Task: Minimize the total *weighted* completion time $\sum_{j=1}^{n} w_j C_j$.

Remarks:

- Unfortunately, already the weighted preemptive problem is NP-hard.

Problem $1|r_j|\sum_j w_j C_j$

Given: As before, but now all jobs j also have a weight $w_j \geq 0$.

Task: Minimize the total *weighted* completion time $\sum_{j=1}^{n} w_j C_j$.

Remarks:

- Unfortunately, already the weighted preemptive problem is NP-hard.
- Thus, instead of preemptive relaxation use LP relaxation:

Problem $1|r_j|\sum_j w_j C_j$

Given: As before, but now all jobs j also have a weight $w_j \geq 0$.

Task: Minimize the total *weighted* completion time $\sum_{j=1}^{n} w_j C_j$.

Remarks:

- Unfortunately, already the weighted preemptive problem is NP-hard.
- Thus, instead of preemptive relaxation use LP relaxation:

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} w_j C_j & \\
\text{s.t.} \quad & C_j \geq r_j + p_j & \text{for all jobs } j = 1, \ldots, n, \\
& \sum_{j \in S} p_j C_j \geq \tfrac{1}{2} p(S)^2 & \text{for all } S \subseteq \{1, \ldots, n\}.
\end{aligned}
$$

Problem $1|r_j|\sum_j w_j C_j$

Given: As before, but now all jobs j also have a weight $w_j \geq 0$.

Task: Minimize the total *weighted* completion time $\sum_{j=1}^{n} w_j C_j$.

Remarks:

- Unfortunately, already the weighted preemptive problem is NP-hard.
- Thus, instead of preemptive relaxation use LP relaxation:

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} w_j C_j \\ \text{s.t.} \quad & C_j \geq r_j + p_j && \text{for all jobs } j = 1, \ldots, n, \\ & \sum_{j \in S} p_j C_j \geq \tfrac{1}{2} p(S)^2 && \text{for all } S \subseteq \{1, \ldots, n\}. \end{aligned}$$

Lemma 4.4

The completion times C_j of a feasible schedule satisfy the LP constraints.

Scheduling in Order of LP Completion Times

Lemma 4.5

Despite the exponential number of constraints, an optimal solution C^* to the LP relaxation can be computed in polynomial time.

Scheduling in Order of LP Completion Times

Lemma 4.5

Despite the exponential number of constraints, an optimal solution C^* to the LP relaxation can be computed in polynomial time.

Proof:... $\square$

Scheduling in Order of LP Completion Times

Lemma 4.5

Despite the exponential number of constraints, an optimal solution C^* to the LP relaxation can be computed in polynomial time.

Proof:… □

Algorithm

1. compute optimal solution C^* to the LP relaxation;

Scheduling in Order of LP Completion Times

Lemma 4.5

Despite the exponential number of constraints, an optimal solution C^* to the LP relaxation can be computed in polynomial time.

Proof:... □

Algorithm

1. compute optimal solution C^* to the LP relaxation;
2. sort jobs such that $C_1^* \leq C_2^* \leq \cdots \leq C_n^*$;

Scheduling in Order of LP Completion Times

Lemma 4.5

Despite the exponential number of constraints, an optimal solution C^* to the LP relaxation can be computed in polynomial time.

Proof:... □

Algorithm

1. compute optimal solution C^* to the LP relaxation;
2. sort jobs such that $C_1^* \leq C_2^* \leq \cdots \leq C_n^*$;
3. schedule all jobs nonpreemptively and as early as possible in this order;

Scheduling in Order of LP Completion Times

Lemma 4.5

Despite the exponential number of constraints, an optimal solution C^* to the LP relaxation can be computed in polynomial time.

Proof:… □

Algorithm

1. compute optimal solution C^* to the LP relaxation;
2. sort jobs such that $C_1^* \leq C_2^* \leq \cdots \leq C_n^*$;
3. schedule all jobs nonpreemptively and as early as possible in this order;

Theorem 4.6

The algorithm above is a 3-approximation algorithm.

Scheduling in Order of LP Completion Times

Lemma 4.5

Despite the exponential number of constraints, an optimal solution C^* to the LP relaxation can be computed in polynomial time.

Proof:... □

Algorithm

1. compute optimal solution C^* to the LP relaxation;
2. sort jobs such that $C_1^* \leq C_2^* \leq \cdots \leq C_n^*$;
3. schedule all jobs nonpreemptively and as early as possible in this order;

Theorem 4.6

The algorithm above is a 3-approximation algorithm.

Proof:... □

Outline

Prize-Collecting Steiner Tree Problem

Given: Graph $G = (V, E)$, root node $r \in V$, edge costs $c_e \geq 0$, $e \in E$, and penalties $\pi_i \geq 0$, $i \in V$.

Prize-Collecting Steiner Tree Problem

Given: Graph $G = (V, E)$, root node $r \in V$, edge costs $c_e \geq 0$, $e \in E$, and penalties $\pi_i \geq 0$, $i \in V$.

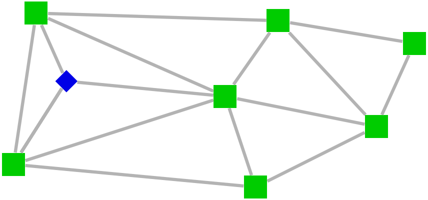
Task: Find subtree T containing root r minimizing $\displaystyle\sum_{e \in E(T)} c_e + \sum_{i \in V \setminus V(T)} \pi_i$.

Prize-Collecting Steiner Tree Problem

Given: Graph $G = (V, E)$, root node $r \in V$, edge costs $c_e \geq 0$, $e \in E$, and penalties $\pi_i \geq 0$, $i \in V$.

Task: Find subtree T containing root r minimizing $\displaystyle\sum_{e \in E(T)} c_e + \sum_{i \in V \setminus V(T)} \pi_i$.

Example:

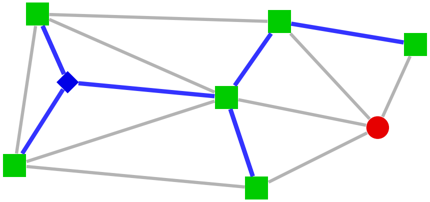


Prize-Collecting Steiner Tree Problem

Given: Graph $G = (V, E)$, root node $r \in V$, edge costs $c_e \geq 0$, $e \in E$, and penalties $\pi_i \geq 0$, $i \in V$.

Task: Find subtree T containing root r minimizing $\sum_{e \in E(T)} c_e + \sum_{i \in V \setminus V(T)} \pi_i$.

Example:



◆ Root node
■ Connected Terminal
● Disonnected Terminal

COST = $c($ —— $) + \pi($ ● $)$

Remark: The Steiner Tree Problem is a special case with $\pi_i = 0$ for all non-terminals and $\pi_i = \infty$ for terminals i.

Prize-Collecting Steiner Tree Problem

Given: Graph $G = (V, E)$, root node $r \in V$, edge costs $c_e \geq 0$, $e \in E$, and penalties $\pi_i \geq 0$, $i \in V$.

Task: Find subtree T containing root r minimizing $\sum_{e \in E(T)} c_e + \sum_{i \in V \setminus V(T)} \pi_i$.

Remark: The Steiner Tree Problem is a special case with $\pi_i = 0$ for all non-terminals and $\pi_i = \infty$ for terminals i.

IP formulation:

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e \cdot x_e + \sum_{i \in V} \pi_i \cdot (1 - y_i) & \\
\text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geq \max_{i \in S} y_i & \text{for all } S \subseteq V \setminus \{r\}, \\
& y_r = 1, & \\
& x_e, y_i \in \{0, 1\} & \text{for all } e \in E, i \in V.
\end{aligned}
$$

Prize-Collecting Steiner Tree Problem

Given: Graph $G = (V, E)$, root node $r \in V$, edge costs $c_e \geq 0$, $e \in E$, and penalties $\pi_i \geq 0$, $i \in V$.

Task: Find subtree T containing root r minimizing $\sum_{e \in E(T)} c_e + \sum_{i \in V \setminus V(T)} \pi_i$.

Remark: The Steiner Tree Problem is a special case with $\pi_i = 0$ for all non-terminals and $\pi_i = \infty$ for terminals i.

IP formulation:

$$
\begin{aligned}
\min \quad & \sum_{e \in E} c_e \cdot x_e + \sum_{i \in V} \pi_i \cdot (1 - y_i) & \\
\text{s.t.} \quad & \sum_{e \in \delta(S)} x_e \geq \max_{i \in S} y_i & \text{for all } S \subseteq V \setminus \{r\}, \\
& y_r = 1, & \\
& x_e, y_i \in \{0, 1\} & \text{for all } e \in E, i \in V.
\end{aligned}
$$

LP relaxation: $x_e \geq 0$ for all $e \in E$ and $y_i \leq 1$ for all $i \in V$.

Deterministic LP Rounding Algorithm

Let $0 \leq \alpha < 1$.

Deterministic LP Rounding Algorithm

Let $0 \leq \alpha < 1$.

1. compute optimal LP solution (x^*, y^*) in polytime with ellipsoid algo;

Deterministic LP Rounding Algorithm

Let $0 \leq \alpha < 1$.

1. compute optimal LP solution (x^*, y^*) in polytime with ellipsoid algo;
2. set $U := \{i \in V \mid y_i^* \geq \alpha\}$;

Deterministic LP Rounding Algorithm

Let $0 \leq \alpha < 1$.

1. compute optimal LP solution (x^*, y^*) in polytime with ellipsoid algo;
2. set $U := \{i \in V \mid y_i^* \geq \alpha\}$;
3. Find Steiner tree T on terminals U using some primal-dual algorithm.

Deterministic LP Rounding Algorithm

Let $0 \leq \alpha < 1$.

1. compute optimal LP solution (x^*, y^*) in polytime with ellipsoid algo;
2. set $U := \{i \in V \mid y_i^* \geq \alpha\}$;
3. Find Steiner tree T on terminals U using some primal-dual algorithm. We will prove the following lemma later, in an exercise:

Lemma 4.7

There is a primal-dual algorithm that returns a Steiner tree T on terminals U with cost at most $\frac{2}{\alpha} \sum_{e \in E} c_e \cdot x_e^*$.

Deterministic LP Rounding Algorithm

Let $0 \leq \alpha < 1$.

1. compute optimal LP solution (x^*, y^*) in polytime with ellipsoid algo;
2. set $U := \{i \in V \mid y_i^* \geq \alpha\}$;
3. Find Steiner tree T on terminals U using some primal-dual algorithm. We will prove the following lemma later, in an exercise:

Lemma 4.7

There is a primal-dual algorithm that returns a Steiner tree T on terminals U with cost at most $\frac{2}{\alpha}\sum_{e\in E} c_e \cdot x_e^*$.

Theorem 4.8

For $\alpha = 2/3$ the cost of the solution returned by the algorithm is

$$c\big(E(T)\big) + \pi\big(V \setminus V(T)\big) \leq \frac{2}{\alpha}\sum_{e\in E} c_e \cdot x_e^* + \frac{1}{1-\alpha}\sum_{i\in V} \pi_i \cdot (1 - y_i^*) \leq 3 \cdot \mathtt{OPT} \quad .$$

Outline

Uncapacitated Facility Location Problem

Given: Set of facilities F with opening costs $f_i \geq 0$, $i \in F$;
set of clients D with connection costs $c_{ij} \geq 0$, $i \in F$, $j \in D$.

Uncapacitated Facility Location Problem

Given: Set of facilities F with opening costs $f_i \geq 0$, $i \in F$;
set of clients D with connection costs $c_{ij} \geq 0$, $i \in F$, $j \in D$.

Task: Choose $F' \subseteq F$ and assign each client to nearest facility in F'.

Uncapacitated Facility Location Problem

Given: Set of facilities F with opening costs $f_i \geq 0$, $i \in F$;
set of clients D with connection costs $c_{ij} \geq 0$, $i \in F$, $j \in D$.

Task: Choose $F' \subseteq F$ and assign each client to nearest facility in F'.

Objective: Minimize $\sum_{i \in F'} f_i + \sum_{j \in D} \min_{i \in F'} c_{ij}$.

Uncapacitated Facility Location Problem

Given: Set of facilities F with opening costs $f_i \geq 0$, $i \in F$;
set of clients D with connection costs $c_{ij} \geq 0$, $i \in F$, $j \in D$.

Task: Choose $F' \subseteq F$ and assign each client to nearest facility in F'.

Objective: Minimize $\sum_{i \in F'} f_i + \sum_{j \in D} \min_{i \in F'} c_{ij}$.

Remarks:

- This is a generalization of the Set Cover Problem.

Uncapacitated Facility Location Problem

Given: Set of facilities F with opening costs $f_i \geq 0$, $i \in F$;
set of clients D with connection costs $c_{ij} \geq 0$, $i \in F$, $j \in D$.

Task: Choose $F' \subseteq F$ and assign each client to nearest facility in F'.

Objective: Minimize $\sum_{i \in F'} f_i + \sum_{j \in D} \min_{i \in F'} c_{ij}$.

Remarks:

- This is a generalization of the Set Cover Problem.
- In the following, we consider the special case with metric costs c_{ij}.

Uncapacitated Facility Location Problem

Given: Set of facilities F with opening costs $f_i \geq 0$, $i \in F$;
set of clients D with connection costs $c_{ij} \geq 0$, $i \in F, j \in D$.

Task: Choose $F' \subseteq F$ and assign each client to nearest facility in F'.

Objective: Minimize $\sum_{i \in F'} f_i + \sum_{j \in D} \min_{i \in F'} c_{ij}$.

Remarks:

- This is a generalization of the Set Cover Problem.
- In the following, we consider the special case with metric costs c_{ij}.

IP formulation:

$$\begin{aligned}
\min_{x_{ij}, y_i \in \{0,1\}} \quad & \sum_{i \in F} f_i \cdot y_i + \sum_{i \in F, j \in D} c_{ij} \cdot x_{ij} & \\
\text{s.t.} \quad & \sum_{i \in F} x_{ij} = 1 & \text{for all } j \in D, \\
& y_i - x_{ij} \geq 0 & \text{for all } i \in F, j \in D.
\end{aligned}$$

LP Relaxation and Dual LP

$$
\begin{aligned}
\min \quad & \sum_{i \in F} f_i \cdot y_i + \sum_{i \in F, j \in D} c_{ij} \cdot x_{ij} & \\
\text{s.t.} \quad & \sum_{i \in F} x_{ij} = 1 & \text{for all } j \in D, \\
& y_i - x_{ij} \geq 0 & \text{for all } i \in F, j \in D, \\
& x_{ij}, y_i \geq 0 & \text{for all } i \in F, j \in D.
\end{aligned}
$$

LP Relaxation and Dual LP

$$
\begin{aligned}
\min \quad & \sum_{i\in F} f_i \cdot y_i + \sum_{i\in F, j\in D} c_{ij} \cdot x_{ij} \\
\text{s.t.} \quad & \sum_{i\in F} x_{ij} = 1 && \text{for all } j \in D, \\
& y_i - x_{ij} \geq 0 && \text{for all } i \in F, j \in D, \\
& x_{ij}, y_i \geq 0 && \text{for all } i \in F, j \in D.
\end{aligned}
$$

dual LP:

$$
\begin{aligned}
\max_{v_j, w_{ij} \geq 0} \quad & \sum_{j\in D} v_j \\
\text{s.t.} \quad & \sum_{j\in D} w_{ij} \leq f_i && \text{for all } i \in F, \\
& v_j - w_{ij} \leq c_{ij} && \text{for all } i \in F, j \in D.
\end{aligned}
$$

LP Relaxation and Dual LP

$$
\begin{aligned}
\min \quad & \sum_{i \in F} f_i \cdot y_i + \sum_{i \in F, j \in D} c_{ij} \cdot x_{ij} \\
\text{s.t.} \quad & \sum_{i \in F} x_{ij} = 1 && \text{for all } j \in D, \\
& y_i - x_{ij} \geq 0 && \text{for all } i \in F, j \in D, \\
& x_{ij}, y_i \geq 0 && \text{for all } i \in F, j \in D.
\end{aligned}
$$

dual LP:

$$
\begin{aligned}
\max_{v_j, w_{ij} \geq 0} \quad & \sum_{j \in D} v_j \\
\text{s.t.} \quad & \sum_{j \in D} w_{ij} \leq f_i && \text{for all } i \in F, \\
& v_j - w_{ij} \leq c_{ij} && \text{for all } i \in F, j \in D.
\end{aligned}
$$

Interpretation of the dual LP:

- v_j is the total amount that client j wants to pay for being served.

LP Relaxation and Dual LP

$$\begin{aligned}
\min \quad & \sum_{i \in F} f_i \cdot y_i + \sum_{i \in F, j \in D} c_{ij} \cdot x_{ij} \\
\text{s.t.} \quad & \sum_{i \in F} x_{ij} = 1 && \text{for all } j \in D, \\
& y_i - x_{ij} \geq 0 && \text{for all } i \in F, j \in D, \\
& x_{ij}, y_i \geq 0 && \text{for all } i \in F, j \in D.
\end{aligned}$$

dual LP:

$$\begin{aligned}
\max_{v_j, w_{ij} \geq 0} \quad & \sum_{j \in D} v_j \\
\text{s.t.} \quad & \sum_{j \in D} w_{ij} \leq f_i && \text{for all } i \in F, \\
& v_j - w_{ij} \leq c_{ij} && \text{for all } i \in F, j \in D.
\end{aligned}$$

Interpretation of the dual LP:

- v_j is the total amount that client j wants to pay for being served.
- client j might contribute w_{ij} to facility i for being connected to i.

Structure of Optimal LP Solution

Let (x^*, y^*) and (v^*, w^*) be optimal solutions to the primal and dual LP, respectively.

Structure of Optimal LP Solution

Let (x^*, y^*) and (v^*, w^*) be optimal solutions to the primal and dual LP, respectively.

Notation:

- Facility i *neighbors* client j if $x_{ij}^* > 0$; $N(j) := \{i \in F \mid x_{ij}^* > 0\}$.
- $N^2(j) := \{\ell \in D \mid \text{client } \ell \text{ neighbors some facility } i \in N(j)\}$.

Structure of Optimal LP Solution

Let (x^*, y^*) and (v^*, w^*) be optimal solutions to the primal and dual LP, respectively.

Notation:

- Facility i *neighbors* client j if $x^*_{ij} > 0$; $N(j) := \{i \in F \mid x^*_{ij} > 0\}$.
- $N^2(j) := \{\ell \in D \mid \text{client } \ell \text{ neighbors some facility } i \in N(j)\}$.

Lemma 4.9

If clients $j_1, \ldots, j_k$ have disjoint neighborhoods $N(j_1), \ldots, N(j_k)$, then opening cheapest facility in each neighborhood costs $\leq \sum_{i \in F} f_i \cdot y^*_i \leq \texttt{OPT}$.

Structure of Optimal LP Solution

Let (x^*, y^*) and (v^*, w^*) be optimal solutions to the primal and dual LP, respectively.

Notation:

- Facility i *neighbors* client j if $x_{ij}^* > 0$; $N(j) := \{i \in F \mid x_{ij}^* > 0\}$.
- $N^2(j) := \{\ell \in D \mid \text{client } \ell \text{ neighbors some facility } i \in N(j)\}$.

Lemma 4.9

If clients $j_1, \dots, j_k$ have disjoint neighborhoods $N(j_1), \dots, N(j_k)$, then opening cheapest facility in each neighborhood costs $\leq \sum_{i \in F} f_i \cdot y_i^* \leq \mathtt{OPT}$.

Lemma 4.10

For each client j, $v_j^* \geq c_{ij}$ for all $i \in N(j)$.

Structure of Optimal LP Solution

Let (x^*, y^*) and (v^*, w^*) be optimal solutions to the primal and dual LP, respectively.

Notation:

- Facility i *neighbors* client j if $x_{ij}^* > 0$; $N(j) := \{i \in F \mid x_{ij}^* > 0\}$.
- $N^2(j) := \{\ell \in D \mid \text{client } \ell \text{ neighbors some facility } i \in N(j)\}$.

Lemma 4.9

If clients $j_1, \ldots, j_k$ have disjoint neighborhoods $N(j_1), \ldots, N(j_k)$, then opening cheapest facility in each neighborhood costs $\leq \sum_{i \in F} f_i \cdot y_i^* \leq \texttt{OPT}$.

Lemma 4.10

For each client j, $v_j^* \geq c_{ij}$ for all $i \in N(j)$.

Proofs: … □

Deterministic LP Rounding Algorithm

```
compute optimal LP solutions (x*, y*) and (v*, w*);
while D ≠ ∅
      choose j := argmin_{j'∈D} v*_{j'} and i := argmin_{i'∈N(j)} f_{i'};
      assign all unassigned clients in N²(j) to facility i;
      set D := D \ N²(j);
```

Deterministic LP Rounding Algorithm

1. compute optimal LP solutions (x^*, y^*) and (v^*, w^*);
2. while $D \neq \emptyset$
3. choose $j := \text{argmin}_{j' \in D} v^*_{j'}$ and $i := \text{argmin}_{i' \in N(j)} f_{i'}$;
4. assign all unassigned clients in $N^2(j)$ to facility i;
5. set $D := D \setminus N^2(j)$;

Theorem 4.11

The algorithm above is a 4-approximation algorithm.

Deterministic LP Rounding Algorithm

1. compute optimal LP solutions (x^*, y^*) and (v^*, w^*);
2. while $D \neq \emptyset$
3. choose $j := \text{argmin}_{j' \in D} v^*_{j'}$ and $i := \text{argmin}_{i' \in N(j)} f_{i'}$;
4. assign all unassigned clients in $N^2(j)$ to facility i;
5. set $D := D \setminus N^2(j)$;

Theorem 4.11

The algorithm above is a 4-approximation algorithm.

Proof:... □

Deterministic LP Rounding Algorithm

1. compute optimal LP solutions (x^*, y^*) and (v^*, w^*);
2. while $D \neq \emptyset$
3. choose $j := \operatorname{argmin}_{j' \in D} v^*_{j'}$ and $i := \operatorname{argmin}_{i' \in N(j)} f_{i'}$;
4. assign all unassigned clients in $N^2(j)$ to facility i;
5. set $D := D \setminus N^2(j)$;

Theorem 4.11

The algorithm above is a 4-approximation algorithm.

Proof:… □

We finally mention the following non-approximability result without proof.

Theorem 4.12

There is no α-approximation algorithm for the metric uncapacitated facility location problem with $\alpha < 1.463$ unless each problem in NP has an $O(n^{O(\log \log n)})$ time algorithm.

Outline

Bin Packing Revisited

In the previous chapter we showed how to find a solution to instance I with at most $(1+\varepsilon)\texttt{OPT}(I)+1$ bins in polynomial time.

Bin Packing Revisited

In the previous chapter we showed how to find a solution to instance I with at most $(1+\varepsilon)\texttt{OPT}(I)+1$ bins in polynomial time.

Goal: Use at most $\texttt{OPT}(I) + O\big(\log^2 \texttt{OPT}(I)\big)$ bins!
(Karmarkar & Karp, 1982)

Bin Packing Revisited

In the previous chapter we showed how to find a solution to instance I with at most $(1+\varepsilon)\texttt{OPT}(I)+1$ bins in polynomial time.

Goal: Use at most $\texttt{OPT}(I)+O\left(\log^2 \texttt{OPT}(I)\right)$ bins!
(Karmarkar & Karp, 1982)

Ingredients:

- Replace dynamic program with integer program plus LP rounding.
- Improved grouping scheme.
- Recursive application of two previous ingredients.

Bin Packing Revisited

In the previous chapter we showed how to find a solution to instance I with at most $(1+\varepsilon)\mathtt{OPT}(I) + 1$ bins in polynomial time.

Goal: Use at most $\mathtt{OPT}(I) + O\big(\log^2 \mathtt{OPT}(I)\big)$ bins!
(Karmarkar & Karp, 1982)

Ingredients:

- Replace dynamic program with integer program plus LP rounding.
- Improved grouping scheme.
- Recursive application of two previous ingredients.

Notice:
By Lemma 3.14 we can assume that all items have size at least $1/\mathrm{SIZE}(I)$.

Configuration Integer Program for Bin Packing

- let $s_1 > s_2 > \cdots > s_m$ denote the different item sizes;
- for $i = 1, \ldots, m$, let b_i denote the number of items of size s_i;

Configuration Integer Program for Bin Packing

- let $s_1 > s_2 > \cdots > s_m$ denote the different item sizes;
- for $i = 1, \ldots, m$, let b_i denote the number of items of size s_i;
- an m-tuple $(t_1, \ldots, t_m) \in \mathbb{Z}^m_{\geq 0}$ is a configuration if $\sum_{i=1}^{m} t_i \cdot s_i \leq 1$;

Configuration Integer Program for Bin Packing

- let $s_1 > s_2 > \cdots > s_m$ denote the different item sizes;
- for $i = 1, \ldots, m$, let b_i denote the number of items of size s_i;
- an m-tuple $(t_1, \ldots, t_m) \in \mathbb{Z}^m_{\geq 0}$ is a configuration if $\sum_{i=1}^{m} t_i \cdot s_i \leq 1$;
- let $T_1, \ldots, T_N$ be a complete enumeration of all configurations and denote by t_{ij} the multiplicity of item i in configuration T_j;

Configuration Integer Program for Bin Packing

- let $s_1 > s_2 > \cdots > s_m$ denote the different item sizes;
- for $i = 1, \ldots, m$, let b_i denote the number of items of size s_i;
- an m-tuple $(t_1, \ldots, t_m) \in \mathbb{Z}^m_{\geq 0}$ is a configuration if $\sum_{i=1}^{m} t_i \cdot s_i \leq 1$;
- let $T_1, \ldots, T_N$ be a complete enumeration of all configurations and denote by t_{ij} the multiplicity of item i in configuration T_j;
- for $j = 1, \ldots, N$, the integer variable x_j denotes the number of bins that shall be packed according to configuration T_j:

Configuration Integer Program for Bin Packing

- let $s_1 > s_2 > \cdots > s_m$ denote the different item sizes;
- for $i = 1, \ldots, m$, let b_i denote the number of items of size s_i;
- an m-tuple $(t_1, \ldots, t_m) \in \mathbb{Z}^m_{\geq 0}$ is a configuration if $\sum_{i=1}^{m} t_i \cdot s_i \leq 1$;
- let $T_1, \ldots, T_N$ be a complete enumeration of all configurations and denote by t_{ij} the multiplicity of item i in configuration T_j;
- for $j = 1, \ldots, N$, the integer variable x_j denotes the number of bins that shall be packed according to configuration T_j:

$$
\begin{array}{lll}
\min & \sum_{j=1}^{N} x_j & \\
\text{s.t.} & \sum_{j=1}^{N} t_{ij} \cdot x_j \geq b_i & \text{for all } i = 1, \ldots, m, \\
& x_j \in \mathbb{Z}_{\geq 0} & \text{for all } j = 1, \ldots, N.
\end{array}
$$

Configuration LP and its Dual

Primal:

$$\begin{aligned} \min \quad & \sum_{j=1}^{N} x_j \\ \text{s.t.} \quad & \sum_{j=1}^{N} t_{ij} \cdot x_j \geq b_i && \text{for all } i = 1, \ldots, m, \\ & x_j \geq 0 && \text{for all } j = 1, \ldots, N. \end{aligned}$$

Configuration LP and its Dual

Primal:

$$\min \sum_{j=1}^{N} x_j$$

$$\text{s.t.} \quad \sum_{j=1}^{N} t_{ij} \cdot x_j \geq b_i \qquad \text{for all } i = 1, \ldots, m,$$

$$x_j \geq 0 \qquad \text{for all } j = 1, \ldots, N.$$

Dual:

$$\max \sum_{i=1}^{m} b_i \cdot y_i$$

$$\text{s.t.} \quad \sum_{i=1}^{m} t_{ij} \cdot y_i \leq 1 \qquad \text{for all } j = 1, \ldots, N,$$

$$y_i \geq 0 \qquad \text{for all } i = 1, \ldots, m.$$

Configuration LP and its Dual

Primal:

$$\min \quad \sum_{j=1}^{N} x_j$$

$$\text{s.t.} \quad \sum_{j=1}^{N} t_{ij} \cdot x_j \geq b_i \qquad \text{for all } i = 1, \ldots, m,$$

$$x_j \geq 0 \qquad \text{for all } j = 1, \ldots, N.$$

Dual:

$$\max \quad \sum_{i=1}^{m} b_i \cdot y_i$$

$$\text{s.t.} \quad \sum_{i=1}^{m} t_{ij} \cdot y_i \leq 1 \qquad \text{for all } j = 1, \ldots, N,$$

$$y_i \geq 0 \qquad \text{for all } i = 1, \ldots, m.$$

Notice: $\text{SIZE}(I) \leq \texttt{OPT}_{LP}(I) \leq \texttt{OPT}(I)$

Solving the Configuration LP Approximately

- Configuration LP suffers from exponentially many variables.

Solving the Configuration LP Approximately

- Configuration LP suffers from exponentially many variables.
- Dual separation problem is Knapsack Problem and thus NP-hard.

Solving the Configuration LP Approximately

- Configuration LP suffers from exponentially many variables.
- Dual separation problem is Knapsack Problem and thus *NP*-hard.
- Remember: optimization and separation are equally difficult.

Solving the Configuration LP Approximately

- Configuration LP suffers from exponentially many variables.
- Dual separation problem is Knapsack Problem and thus *NP*-hard.
- Remember: optimization and separation are equally difficult.
- Therefore, it is *NP*-hard to solve the Configuration LP to optimality.

Solving the Configuration LP Approximately

- Configuration LP suffers from exponentially many variables.
- Dual separation problem is Knapsack Problem and thus *NP*-hard.
- Remember: optimization and separation are equally difficult.
- Therefore, it is *NP*-hard to solve the Configuration LP to optimality.

Theorem 4.13

An LP solution of value at most $\mathrm{OPT}_{LP}(I) + 1$ can be computed in polynomial time.

Solving the Configuration LP Approximately

- Configuration LP suffers from exponentially many variables.
- Dual separation problem is Knapsack Problem and thus *NP*-hard.
- Remember: optimization and separation are equally difficult.
- Therefore, it is *NP*-hard to solve the Configuration LP to optimality.

Theorem 4.13

An LP solution of value at most $\mathrm{OPT}_{LP}(I) + 1$ can be computed in polynomial time.

Proof:… □

Proof of Theorem 4.13

Main idea: Use FPTAS for Knapsack Problem as approximate separation routine within ellipsoid method.

Proof of Theorem 4.13

Main idea: Use FPTAS for Knapsack Problem as approximate separation routine within ellipsoid method. This yields optimal solution y^* to

perturbed dual:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} b_i \cdot y_i \\ \text{s.t.} \quad & \sum_{i=1}^{m} t_{ij} \cdot y_i \leq \delta_j \qquad \text{for all } j = 1, \ldots, N, \\ & y_i \geq 0 \qquad \text{for all } i = 1, \ldots, m, \end{aligned}$$

with $\delta_j \in \{1, 1+\varepsilon\}$ and $|\{j \mid \delta_j = 1\}|$ polynomially bounded.

Proof of Theorem 4.13

Main idea: Use FPTAS for Knapsack Problem as approximate separation routine within ellipsoid method. This yields optimal solution y^* to

perturbed dual:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} b_i \cdot y_i & \\ \text{s.t.} \quad & \sum_{i=1}^{m} t_{ij} \cdot y_i \leq \delta_j & \text{for all } j = 1, \ldots, N, \\ & y_i \geq 0 & \text{for all } i = 1, \ldots, m, \end{aligned}$$

with $\delta_j \in \{1, 1+\varepsilon\}$ and $|\{j \mid \delta_j = 1\}|$ polynomially bounded.

Since $y^*/(1+\varepsilon)$ is feasible dual solution, $\sum_{i=1}^{m} b_i \cdot y_i^* \leq (1+\varepsilon)\mathrm{OPT}_{LP}$.

Proof of Theorem 4.13

Main idea: Use FPTAS for Knapsack Problem as approximate separation routine within ellipsoid method. This yields optimal solution y^* to

perturbed dual:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} b_i \cdot y_i \\ \text{s.t.} \quad & \sum_{i=1}^{m} t_{ij} \cdot y_i \leq \delta_j && \text{for all } j = 1, \ldots, N, \\ & y_i \geq 0 && \text{for all } i = 1, \ldots, m, \end{aligned}$$

with $\delta_j \in \{1, 1+\varepsilon\}$ and $|\{j \mid \delta_j = 1\}|$ polynomially bounded.

Since $y^*/(1+\varepsilon)$ is feasible dual solution, $\sum_{i=1}^{m} b_i \cdot y_i^* \leq (1+\varepsilon)\text{OPT}_{LP}$.

Moreover, for $J := \{j \mid \delta_j = 1\}$, vector y^* is optimal solution to

reduced dual:

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} b_i \cdot y_i \\ \text{s.t.} \quad & \sum_{i=1}^{m} t_{ij} \cdot y_i \leq 1 && \text{for all } j \text{ with } \delta_j = 1, \\ & y_i \geq 0 && \text{for all } i = 1, \ldots, m. \end{aligned}$$

Proof of Theorem 4.13 (Cont.)

Consider the corresponding

reduced primal:

$$
\begin{aligned}
\min \quad & \sum_{j \in J} x_j \\
\text{s.t.} \quad & \sum_{j \in J} t_{ij} \cdot x_j \geq b_i && \text{for all } i = 1, \dots, m, \\
& x_j \geq 0 && \text{for all } j \in J.
\end{aligned}
$$

Proof of Theorem 4.13 (Cont.)

Consider the corresponding

reduced primal:

$$\begin{aligned} \min \quad & \sum_{j \in J} x_j \\ \text{s.t.} \quad & \sum_{j \in J} t_{ij} \cdot x_j \geq b_i && \text{for all } i = 1, \ldots, m, \\ & x_j \geq 0 && \text{for all } j \in J. \end{aligned}$$

It has polynomial size and optimal solution value at most $(1 + \varepsilon)\mathrm{OPT}_{LP}$.

Proof of Theorem 4.13 (Cont.)

Consider the corresponding

reduced primal:

$$\begin{aligned} \min \quad & \sum_{j \in J} x_j \\ \text{s.t.} \quad & \sum_{j \in J} t_{ij} \cdot x_j \geq b_i && \text{for all } i = 1, \ldots, m, \\ & x_j \geq 0 && \text{for all } j \in J. \end{aligned}$$

It has polynomial size and optimal solution value at most $(1 + \varepsilon)\mathrm{OPT}_{LP}$.

Choose $\varepsilon := 1/n$ such that $(1 + \varepsilon)\mathrm{OPT}_{LP} \leq \mathrm{OPT}_{LP} + \varepsilon n \leq \mathrm{OPT}_{LP} + 1$.

Proof of Theorem 4.13 (Cont.)

Consider the corresponding

reduced primal:

$$\begin{aligned} \min \quad & \sum_{j \in J} x_j & \\ \text{s.t.} \quad & \sum_{j \in J} t_{ij} \cdot x_j \geq b_i & \text{for all } i = 1, \ldots, m, \\ & x_j \geq 0 & \text{for all } j \in J. \end{aligned}$$

It has polynomial size and optimal solution value at most $(1 + \varepsilon)\mathrm{OPT}_{LP}$.

Choose $\varepsilon := 1/n$ such that $(1 + \varepsilon)\mathrm{OPT}_{LP} \leq \mathrm{OPT}_{LP} + \varepsilon n \leq \mathrm{OPT}_{LP} + 1$.

Reduced primal and its optimal solution $\bar{y}$ can be computed in polynomial time (FPTAS for Knapsack!).

Proof of Theorem 4.13 (Cont.)

Consider the corresponding

reduced primal:

$$\begin{aligned} \min \quad & \sum_{j \in J} x_j & \\ \text{s.t.} \quad & \sum_{j \in J} t_{ij} \cdot x_j \geq b_i & \text{for all } i = 1, \ldots, m, \\ & x_j \geq 0 & \text{for all } j \in J. \end{aligned}$$

It has polynomial size and optimal solution value at most $(1+\varepsilon)\mathrm{OPT}_{LP}$.

Choose $\varepsilon := 1/n$ such that $(1+\varepsilon)\mathrm{OPT}_{LP} \leq \mathrm{OPT}_{LP} + \varepsilon n \leq \mathrm{OPT}_{LP} + 1$.

Reduced primal and its optimal solution $\bar{y}$ can be computed in polynomial time (FPTAS for Knapsack!).

$\bar{y}$ is feasible solution to original primal LP of value at most $\mathrm{OPT}_{LP} + 1$. □

Harmonic Grouping Scheme and Rounding

Grouping

- consider items in order of non-increasing size;
- open a group and start putting items in current group, one at a time;
- close current group if its total size is at least 2 and start new group;

Harmonic Grouping Scheme and Rounding

Grouping

- consider items in order of non-increasing size;
- open a group and start putting items in current group, one at a time;
- close current group if its total size is at least 2 and start new group;

Let $r :=$ number of groups; let G_i denote ith group; $n_i := |G_i|$.

Harmonic Grouping Scheme and Rounding

Grouping

- consider items in order of non-increasing size;
- open a group and start putting items in current group, one at a time;
- close current group if its total size is at least 2 and start new group;

Let $r :=$ number of groups; let G_i denote ith group; $n_i := |G_i|$.

Notice that $r \leq \lceil \text{SIZE}(I)/2 \rceil$ and $n_i \geq n_{i-1}$, for $i = 2, \dots, r-1$.

Harmonic Grouping Scheme and Rounding

Grouping

- consider items in order of non-increasing size;
- open a group and start putting items in current group, one at a time;
- close current group if its total size is at least 2 and start new group;

Let $r :=$ number of groups; let G_i denote ith group; $n_i := |G_i|$.

Notice that $r \leq \lceil \text{SIZE}(I)/2 \rceil$ and $n_i \geq n_{i-1}$, for $i = 2, \ldots, r-1$.

Rounding: Construct new instance I' as follows:

- discard items in G_1 and G_r;
- for $i = 2, \ldots, r-1$ discard the $n_i - n_{i-1}$ smallest items in G_i;
- for $i = 2, \ldots, r-1$ round sizes of remaining items in G_i to largest one.

Harmonic Grouping Scheme and Rounding

Grouping

- consider items in order of non-increasing size;
- open a group and start putting items in current group, one at a time;
- close current group if its total size is at least 2 and start new group;

Let $r :=$ number of groups; let G_i denote ith group; $n_i := |G_i|$.

Notice that $r \leq \lceil \text{SIZE}(I)/2 \rceil$ and $n_i \geq n_{i-1}$, for $i = 2, \dots, r-1$.

Rounding: Construct new instance I' as follows:

- discard items in G_1 and G_r;
- for $i = 2, \dots, r-1$ discard the $n_i - n_{i-1}$ smallest items in G_i;
- for $i = 2, \dots, r-1$ round sizes of remaining items in G_i to largest one.

Lemma 4.14

There are at most $\text{SIZE}(I)/2$ distinct item sizes in I'; the total size of all discarded items is $O(\log \text{SIZE}(I))$. □

Recursive Bin Packing Algorithm

BinPack(I)

1. if SIZE(I) < 10 then pack remaining items using First-Fit and stop;

Recursive Bin Packing Algorithm

BinPack(I)

1. if SIZE(I) < 10 then pack remaining items using First-Fit and stop;
2. apply harmonic grouping scheme to create instance I';
3. pack discarded items in $O(\log \text{SIZE}(I))$ bins using First-Fit;

Recursive Bin Packing Algorithm

BinPack(I)

1. if SIZE(I) < 10 then pack remaining items using First-Fit and stop;
2. apply harmonic grouping scheme to create instance I';
3. pack discarded items in $O(\log \text{SIZE}(I))$ bins using First-Fit;
4. compute near-optimal solution x to Configuration LP for instance I';
5. for $j = 1, \ldots, N$ pack $\lfloor x_j \rfloor$ bins in configuration T_j;
6. call the packed items instance I_1 and the remaining items I_2;

Recursive Bin Packing Algorithm

BinPack(I)

1. if SIZE(I) < 10 then pack remaining items using First-Fit and stop;
2. apply harmonic grouping scheme to create instance I';
3. pack discarded items in $O(\log \text{SIZE}(I))$ bins using First-Fit;
4. compute near-optimal solution x to Configuration LP for instance I';
5. for $j = 1, \ldots, N$ pack $\lfloor x_j \rfloor$ bins in configuration T_j;
6. call the packed items instance I_1 and the remaining items I_2;
7. pack I_2 recursively via BinPack(I_2);

Analysis of Algorithm BinPack

Lemma 4.15

$\mathtt{OPT}_{LP}(I_1) + \mathtt{OPT}_{LP}(I_2) \leq \mathtt{OPT}_{LP}(I') \leq \mathtt{OPT}_{LP}(I).$

Analysis of Algorithm BinPack

Lemma 4.15

$\mathtt{OPT}_{LP}(I_1) + \mathtt{OPT}_{LP}(I_2) \leq \mathtt{OPT}_{LP}(I') \leq \mathtt{OPT}_{LP}(I)$.

Proof:… □

Analysis of Algorithm BinPack

Lemma 4.15

$\mathtt{OPT}_{LP}(I_1) + \mathtt{OPT}_{LP}(I_2) \leq \mathtt{OPT}_{LP}(I') \leq \mathtt{OPT}_{LP}(I)$.

Proof:... $\square$

Theorem 4.16 (Karmarkar & Karp, 1982)

Algorithm BinPack runs in polynomial time and finds a solution using at most $\mathtt{OPT}(I) + O\left(\log^2 \mathtt{OPT}(I)\right)$ bins.

Analysis of Algorithm BinPack

Lemma 4.15

$\texttt{OPT}_{LP}(I_1) + \texttt{OPT}_{LP}(I_2) \leq \texttt{OPT}_{LP}(I') \leq \texttt{OPT}_{LP}(I)$.

Proof:… □

Theorem 4.16 (Karmarkar & Karp, 1982)

Algorithm BinPack runs in polynomial time and finds a solution using at most $\texttt{OPT}(I) + O\left(\log^2 \texttt{OPT}(I)\right)$ bins.

Proof:… □

Analysis of Algorithm BinPack

Lemma 4.15

$\mathtt{OPT}_{LP}(I_1) + \mathtt{OPT}_{LP}(I_2) \leq \mathtt{OPT}_{LP}(I') \leq \mathtt{OPT}_{LP}(I)$.

Proof:… □

Theorem 4.16 (Karmarkar & Karp, 1982)

Algorithm BinPack runs in polynomial time and finds a solution using at most $\mathtt{OPT}(I) + O\left(\log^2 \mathtt{OPT}(I)\right)$ bins.

Proof:… □

Theorem 4.17 (Hoberg & Rothvoß, 2015)

A solution using at most $\mathtt{OPT}(I) + O\left(\log \mathtt{OPT}(I)\right)$ bins can be found in polynomial time. □