Theory of Algorithms IV

Approximation Algorithms

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Introduction

A practical solution to the hard (combinatorial optimal) problems is to design algorithms that provide approximate answers.

Introduction

A practical solution to the hard (combinatorial optimal) problems is to design algorithms that provide approximate answers.

An approximate algorithm must be evaluated by comparing the approximate answer it produces and the optimal answer of the problem.

Introduction

A practical solution to the hard (combinatorial optimal) problems is to design algorithms that provide approximate answers.

An approximate algorithm must be evaluated by comparing the approximate answer it produces and the optimal answer of the problem.

There are hard problems for which no reasonable approximation algorithms exist.

Introduction

We will be interested in optimization problems. Clearly if a decision problem is NP hard, then the corresponding optimization problem is also NP hard.

Introduction

A combinatorial optimal problem Π is either a *minimization problem* or a *maximization problem*. It consists of three components:

- A set D_Π of *instances*;
- For each instance $I \in D_\Pi$, there is a finite set $S_\Pi(I)$ of *candidate solutions* for I;
- Associated with each solution $\sigma \in S_\Pi(I)$ to an instance I in D_Π, there is a value $f_\Pi(\sigma)$ called the solution value for σ.

Introduction

A combinatorial optimal problem Π is either a *minimization problem* or a *maximization problem*. It consists of three components:

- A set D_Π of *instances*;
- For each instance $I \in D_\Pi$, there is a finite set $S_\Pi(I)$ of *candidate solutions* for I;
- Associated with each solution $\sigma \in S_\Pi(I)$ to an instance I in D_Π, there is a value $f_\Pi(\sigma)$ called the solution value for σ.

Given an instance $I \in D_\Pi$, let σ^* be the optimal solution in $S_\Pi(I)$. We shall denote by $OPT(I)$ the value $f_\Pi(\sigma^*)$.

Approximation Algorithm

An approximation algorithm A for an optimization problem Π is a (polynomial time) algorithm such that given an instance $I \in D_\Pi$, it outputs some solution $\sigma \in S_\Pi(I)$. We will denote by $A(I)$ the value $f_\Pi(\sigma)$.

The Bin Packing Problem

Given a collection of items of sizes between 0 and 1, it is required to pack these items into the minimum number of bins of unit capacity.

The Bin Packing Problem

Given a collection of items of sizes between 0 and 1, it is required to pack these items into the minimum number of bins of unit capacity.

In this instance, what are D_Π and $S_\Pi(I)$? What is the solution value function?

Difference Bounds

The best one could hope from an approximation algorithm A is that, for all instances I in D_Π, the difference $|A(I) - OPT(I)|$ is no more than a constant, say K. In other words, the algorithm A satisfies

$$\forall I \in D_\Pi . |A(I) - OPT(I)| \leq K$$

Such approximation algorithms are very rare.

Nonexistence of Difference Bounds

Knapsack Problem: Given n items $\{u_1, u_2, \ldots, u_n\}$ with integer values $s_1, s_2, \ldots, s_n$ and integer values $v_1, v_2, \ldots, v_n$, and a knapsack capacity C that is a positive integer, find a subset $S \subseteq \{u_1, u_2, \ldots, u_n\}$ such that $\sum_{u_j \in S} s_j \leq C$ and $\sum_{u_j \in S} v_j$ is maximum.

Nonexistence of Difference Bounds

Suppose A was an approximation algorithm for the Knapsack Problem such that $\forall I \in D_{\Pi}.|A(I) - OPT(I)| \leq K$ for some positive integer K.

Nonexistence of Difference Bounds

Suppose A was an approximation algorithm for the Knapsack Problem such that $\forall I \in D_\Pi . |A(I) - OPT(I)| \leq K$ for some positive integer K.

Given an instance I, we can obtain a new instance I' by multiplying the values of each item of I by $K + 1$. Now $|A(I') - OPT(I')| \leq K$ implies that $|A(I) - OPT(I)| = 0$, meaning that A must be an optimal algorithm.

Nonexistence of Difference Bounds

Suppose A was an approximation algorithm for the Knapsack Problem such that $\forall I \in D_{\Pi}.|A(I) - OPT(I)| \leq K$ for some positive integer K.

Given an instance I, we can obtain a new instance I' by multiplying the values of each item of I by $K + 1$. Now $|A(I') - OPT(I')| \leq K$ implies that $|A(I) - OPT(I)| = 0$, meaning that A must be an optimal algorithm.

This is possible only if $P = NP$.

Relative Performance Bounds

Suppose A is an approximation algorithm to solve a minimization (or maximization) problem Π. Then

- the approximation ratio $R_A(I)$ is defined by

$$R_A(I) = \frac{A(I)}{OPT(I)} \text{ (or } R_A(I) = \frac{OPT(I)}{A(I)}\text{)}$$

- the absolute performance ratio R_A for A is defined by

$$R_A = \inf\{r \geq 1 \mid \forall I \in D_\Pi . R_A(I) \leq r\}$$

- the asymptotic performance ratio R_A^∞ for A is defined by

$$R_A^\infty = \inf\{r \geq 1 \mid \exists N . \forall I \in D_\Pi . (OPT(I) \geq N \Rightarrow R_A(I) \leq r\})$$

Bin Packing Revisited

First Fit (FF). The bins are indexed as $1, 2, \ldots$. All bins are initially empty. The items are considered for packing in the order $u_1, u_2, \ldots$. To pack item u_i, find the least index j such that bin j contains at most $1 - s_i$, and add item u_i to the items packed in bin j.

Bin Packing Revisited

First Fit (FF). The bins are indexed as $1, 2, \ldots$. All bins are initially empty. The items are considered for packing in the order $u_1, u_2, \ldots$. To pack item u_i, find the least index j such that bin j contains at most $1 - s_i$, and add item u_i to the items packed in bin j.

Best Fit (BF). It differs from FF in that the item u_i is packed into bin j such that $s_j \leq 1 - s_i$ with s_j the maximum.

Bin Packing Revisited

First Fit (FF). The bins are indexed as $1, 2, \ldots$. All bins are initially empty. The items are considered for packing in the order $u_1, u_2, \ldots$. To pack item u_i, find the least index j such that bin j contains at most $1 - s_i$, and add item u_i to the items packed in bin j.

Best Fit (BF). It differs from FF in that the item u_i is packed into bin j such that $s_j \leq 1 - s_i$ with s_j the maximum.

First Fit Decreasing (FFD). First order the items by decreasing order of size, and then use FF.

Bin Packing Revisited

First Fit (FF). The bins are indexed as $1, 2, \ldots$. All bins are initially empty. The items are considered for packing in the order $u_1, u_2, \ldots$. To pack item u_i, find the least index j such that bin j contains at most $1 - s_i$, and add item u_i to the items packed in bin j.

Best Fit (BF). It differs from FF in that the item u_i is packed into bin j such that $s_j \leq 1 - s_i$ with s_j the maximum.

First Fit Decreasing (FFD). First order the items by decreasing order of size, and then use FF.

Best Fit Decreasing (BFD). First order the items by decreasing order of size, and then use BF.

Bin Packing Revisited

Let I be an instance of Bin Packing. If $FF(I) > 1$, then

$$FF(I) < \lceil 2 \sum_{i=1}^{n} s_i \rceil \tag{1}$$

and

$$OPT(I) \geq \lceil \sum_{i=1}^{n} s_i \rceil \tag{2}$$

Bin Packing Revisited

Let I be an instance of Bin Packing. If $FF(I) > 1$, then

$$FF(I) < \lceil 2 \sum_{i=1}^{n} s_i \rceil \tag{1}$$

and

$$OPT(I) \geq \lceil \sum_{i=1}^{n} s_i \rceil \tag{2}$$

It follows immediately from (1) and (2) that

$$R_{FF}(I) = \frac{FF(I)}{OPT(I)} < 2$$

Euclidean Traveling SalesPerson

Given a set of n points in the plane, find a tour (circular path) τ on these points of shortest length.

Euclidean Traveling SalesPerson

A heuristics *EST* for constructing ETPS works as follows:

1. Construct a minimum spanning tree T.
2. A multigraph T' is constructed by making two copies of every edge in T.
3. An Eulerian tour τ_e is found in T'. (A Eulerian tour is a cycle that visits each edge exactly once.)
4. A Eulerian tour τ_e is converted to a Hamiltonian tour τ by shortcutting the vertices that have already visited.

Performance Ratio

Let τ^* denote an optimal tour:

- By definition, $length(T) < length(\tau^*)$
- Hence $length(\tau_e) < 2length(\tau^*)$
- By triangle inequality, $length(\tau) < 2length(\tau^*)$
- Conclude that $R_{MST} < 2$.

An Improved Heuristics

An Eulerian graph is a graph in which every vertex has an even degree. An Eulerian tour in a graph is a circuit that traverses every edge exactly once.

An Improved Heuristics

An Eulerian graph is a graph in which every vertex has an even degree. An Eulerian tour in a graph is a circuit that traverses every edge exactly once.

A graph has an Eulerian tour iff it is an Eulerian graph.

An Improved Heuristics

An Eulerian graph is a graph in which every vertex has an even degree. An Eulerian tour in a graph is a circuit that traverses every edge exactly once.

A graph has an Eulerian tour iff it is an Eulerian graph.

There is a polynomial time algorithm to find an Eulerian tour in an Eulerian graph.

An Improved Heuristics

Given a set $V = \{a_1, a_2, \ldots, a_{2k}\}$ of $2k$ points in the plane, a matching for V is a partition of V into k 2-element sets $\{a_{m_i}, a_{n_i}\}$. The weight of such a matching is the sum of the distances $\sum_{1 \leq i \leq k} d(a_{m_i}, a_{n_i})$.

An Improved Heuristics

Given a set $V = \{a_1, a_2, \ldots, a_{2k}\}$ of $2k$ points in the plane, a matching for V is a partition of V into k 2-element sets $\{a_{m_i}, a_{n_i}\}$. The weight of such a matching is the sum of the distances $\sum_{1 \leq i \leq k} d(a_{m_i}, a_{n_i})$.

The minimum weight matching is a matching with minimum such weight.

An Improved Approximation Algorithm

Algorithm ETSPAPPROX
Input: A set S of n points in the plane.
Output: An Eulerian tour τ of S.

1. Construct a minimum spanning tree T of S
2. Identify the set X of odd degree vertices in T
3. Find a minimum weight matching M on X
4. Find a Eulerian tour τ_e in $T \cup M$
5. Traverse τ_e edge by edge and bypass every previously visited vertex.
6. Let τ be the resulting tour

An Improved Approximation Algorithm

Algorithm ETSPAPPROX
Input: A set S of n points in the plane.
Output: An Eulerian tour τ of S.

1. Construct a minimum spanning tree T of S
2. Identify the set X of odd degree vertices in T
3. Find a minimum weight matching M on X
4. Find a Eulerian tour τ_e in $T \cup M$
5. Traverse τ_e edge by edge and bypass every previously visited vertex.
6. Let τ be the resulting tour

The time complexity of ETSPAPPROX is $O(n^3)$.

Performance Ratio

Let τ^* denote an optimal tour: First observe that
$length(M) < \frac{1}{2} length(\tau^*)$.

Performance Ratio

Let τ^* denote an optimal tour: First observe that $length(M) < \frac{1}{2}length(\tau^*)$.

$$
\begin{aligned}
length(\tau) &\leq length(\tau_e) \\
&\leq length(T) + length(M) \\
&< length(\tau^*) + \frac{1}{2}length(\tau^*) \\
&= \frac{3}{2}length(\tau^*)
\end{aligned}
$$

Performance Ratio

Let τ^* denote an optimal tour: First observe that $length(M) < \frac{1}{2} length(\tau^*)$.

$$\begin{aligned} length(\tau) &\leq length(\tau_e) \\ &\leq length(T) + length(M) \\ &< length(\tau^*) + \frac{1}{2} length(\tau^*) \\ &= \frac{3}{2} length(\tau^*) \end{aligned}$$

Hence $R_{ETSPAPPROX} < \frac{3}{2}$.

Hardness Result: the Traveling Salesperson Problem

Theorem

There is no approximation algorithm A *for the problem Traveling Salesperson with* $R_A < \infty$ *unless* $NP = P$.

Hardness Result: the Traveling Salesperson Problem

Proof: Suppose that there was an approximation algorithm A for the problem Traveling Salesperson with $R_A \leq K$. We will show that there would be a polynomial time algorithm for the problem Hamiltonian Cycle.

Hardness Result: the Traveling Salesperson Problem

Proof: Suppose that there was an approximation algorithm A for the problem Traveling Salesperson with $R_A \leq K$. We will show that there would be a polynomial time algorithm for the problem Hamiltonian Cycle.

Let $G = (V, E)$ be an undirected graph. Define an instance I of the Traveling Salesperson problem by defining the distance function as follows:

$$d(u, v) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } (u, v) \in E \\ Kn, & \text{if } (u, v) \notin E \end{cases}$$

If G has a Hamiltonian cycle then $OPT(I) = n$; otherwise $OPT(I) > Kn$.

Hardness Result: the Traveling Salesperson Problem

Proof: Suppose that there was an approximation algorithm A for the problem Traveling Salesperson with $R_A \leq K$. We will show that there would be a polynomial time algorithm for the problem Hamiltonian Cycle.

Let $G = (V, E)$ be an undirected graph. Define an instance I of the Traveling Salesperson problem by defining the distance function as follows:

$$d(u, v) \stackrel{\text{def}}{=} \begin{cases} 1, & \text{if } (u, v) \in E \\ Kn, & \text{if } (u, v) \notin E \end{cases}$$

If G has a Hamiltonian cycle then $OPT(I) = n$; otherwise $OPT(I) > Kn$.

It should be clear from the assumption that I has a Hamiltonian cycle if and only if $A(I) = n$. This is impossible unless $NP = P$.

The Vertex Cover Problem

An intuitive heuristic: Pick up an edge e arbitrarily and add one of its endpoints, say v, to the vertex cover. Next delete e and all other edges incident to v. However the performance ratio of this algorithm is unbounded.

The Vertex Cover Problem

An intuitive heuristic: Pick up an edge e arbitrarily and add one of its endpoints, say v, to the vertex cover. Next delete e and all other edges incident to v. However the performance ratio of this algorithm is unbounded.

The performance ration becomes 2 with a slight modification.

VCOVERAPPROX

Algorithm VCOVERAPPROX
Input: An undirected graph $G = (V, E)$.
Output: A vertex cover C for G.

1. $C \leftarrow \{\}$
2. **while** $E \neq \{\}$
3.   Let $e = (u, v)$ be any edge in E
4. $C \leftarrow C \cup \{u, v\}$
5.   Remove e and all edges incident to u or v from E.
6. **end while**

Performance Ratio

The set C produced by VCOVERAPPROX is obviously a vertex cover. On the other hand no two edges removed in Step 3 share a vertex.

Performance Ratio

The set C produced by VCOVERAPPROX is obviously a vertex cover. On the other hand no two edges removed in Step 3 share a vertex.

It follows that the size of the optimal vertex cover must be at least half the size of C. So the performance ratio is 2.

Quiz: Set Cover

Set Cover

- Input: A set of elements B, sets $S_1, \ldots, S_m \subseteq B$
- Output: A selection of the S_i whose union is B.
- Cost: Number of sets picked.

Quiz: Set Cover

Set Cover

- Input: A set of elements B, sets $S_1, \ldots, S_m \subseteq B$
- Output: A selection of the S_i whose union is B.
- Cost: Number of sets picked.

- Does the greedy solution is a good approximation?

Approximation Scheme

Approximation Scheme

Let Π be an **NP**-hard optimization problem with objective function f_Π. We will say that algorithm $\mathcal{A}$ is an approximation scheme for Π if on input (I, ϵ), where I is an instance of Π and $\epsilon > 0$ is an error parameter, it outputs a solution s such that:

Approximation Scheme

Let Π be an **NP**-hard optimization problem with objective function f_Π. We will say that algorithm $\mathcal{A}$ is an approximation scheme for Π if on input (I, ϵ), where I is an instance of Π and $\epsilon > 0$ is an error parameter, it outputs a solution s such that:

- $f_\Pi(I, s) \leq (1 + \epsilon) \cdot \mathrm{OPT}$ if Π is a minimization problem.

Approximation Scheme

Let Π be an **NP**-hard optimization problem with objective function f_Π. We will say that algorithm $\mathcal{A}$ is an approximation scheme for Π if on input (I, ϵ), where I is an instance of Π and $\epsilon > 0$ is an error parameter, it outputs a solution s such that:

- $f_\Pi(I, s) \leq (1 + \epsilon) \cdot \mathrm{OPT}$ if Π is a minimization problem.
- $f_\Pi(I, s) \geq (1 - \epsilon) \cdot \mathrm{OPT}$ if Π is a maximization problem.

PTAS and FPTAS

PTAS and FPTAS

$\mathcal{A}$ will be said to be a polynomial time approximation scheme, abbreviated PTAS, if for each fixed $\epsilon > 0$, its running time is bounded by a polynomial in the size of instance I.

PTAS and FPTAS

$\mathcal{A}$ will be said to be a polynomial time approximation scheme, abbreviated PTAS, if for each fixed $\epsilon > 0$, its running time is bounded by a polynomial in the size of instance I.

If we require that the running time of $\mathcal{A}$ be bounded by a polynomial in the size of instance I and $1/\epsilon$, then $\mathcal{A}$ will be said to be a fully polynomial approximation scheme, abbreviated FPTAS.

Knapsack: Problem Statement

Given a set $S = \{a_1, \ldots, a_n\}$ of objects, with specified sizes and profits, $\text{size}(a_i) \in \mathbb{Z}^+$ and $\text{profit}(a_i) \in \mathbb{Z}^+$, and a "knapsack capacity" $B \in \mathbb{Z}^+$, find a subset of objects whose total size is bounded by B and total profit is maximized.

An Example

Objects	A	B	C	D	E
Sizes	7	2	9	3	1
Profits	3	2	3	1	2

Knapsack size: B

Greedy is Bad

Greedy is Bad

An obvious algorithm for this problem is to sort the objects by decreasing ratio of profit to size, and then greedily pick objects in this order.

Greedy is Bad

An obvious algorithm for this problem is to sort the objects by decreasing ratio of profit to size, and then greedily pick objects in this order.

It is easy to see that as such this algorithm can be made to perform arbitrarily badly.

Greedy is Bad

An obvious algorithm for this problem is to sort the objects by decreasing ratio of profit to size, and then greedily pick objects in this order.

It is easy to see that as such this algorithm can be made to perform arbitrarily badly.

$$100/1, (100 * B - 1)/B$$

Dynamic Programming

Dynamic Programming

Let P be the profit of the most profitable object, i.e.,

$$P = \max_{a \in S} \text{profit}(a)$$

Dynamic Programming

Let P be the profit of the most profitable object, i.e.,

$$P = \max_{a \in S} \text{profit}(a)$$

Then nP is a trivial upper bound on the profit that can be achieved by any solution.

Dynamic Programming

Let P be the profit of the most profitable object, i.e.,

$$P = \max_{a \in S} \text{profit}(a)$$

Then nP is a trivial upper bound on the profit that can be achieved by any solution.

For each $i \in \{1, \ldots, n\}$ and $p \in \{1, \ldots, nP\}$, let $S_{i,p}$ denote a subset of $\{a_1, \ldots, a_i\}$ whose total profit is exactly p and whose total size is minimized.

Dynamic Programming

$A(i,p)$ denote the size of the set $S_{i,p}$ ($A(i,p) = \infty$ if no such set exists).

Dynamic Programming

$A(i,p)$ denote the size of the set $S_{i,p}$ ($A(i,p) = \infty$ if no such set exists).

$A(1,p)$ is known for every $p \in \{1,\ldots,nP\}$.

Dynamic Programming

$A(i,p)$ denote the size of the set $S_{i,p}$ ($A(i,p) = \infty$ if no such set exists).

$A(1,p)$ is known for every $p \in \{1,\ldots,nP\}$.

The following recurrence helps compute all values $A(i,p)$ in $O(n^2P)$ time:

$$A(i+1,p) = \begin{cases} \min\{A(i,p), \text{size}(a_{i+1}) + A(i,p-\text{profit}(a_{i+1}))\} & \text{if profit}(a_{i+1}) \leq p \\ A(i,p) & \text{otherwise} \end{cases}$$

Dynamic Programming

$A(i,p)$ denote the size of the set $S_{i,p}$ ($A(i,p) = \infty$ if no such set exists).

$A(1,p)$ is known for every $p \in \{1,\ldots,nP\}$.

The following recurrence helps compute all values $A(i,p)$ in $O(n^2P)$ time:

$$A(i+1,p) = \begin{cases} \min\{A(i,p), \mathrm{size}(a_{i+1}) + A(i,p-\mathrm{profit}(a_{i+1}))\} & \text{if } \mathrm{profit}(a_{i+1}) \leq p \\ A(i,p) & \text{otherwise} \end{cases}$$

The maximum profit achievable by objects of total size bounded by B is $\max\{p \mid A(n,p) \leq B\}$.

An Example

Objects	A	B	C	D	E
Sizes	7	2	9	3	1
Profits	3	2	3	1	2

Knapsack size: B

An Example

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	∞	∞	7	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	∞	2	7	∞	9	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
3	∞	2	7	∞	9	16	∞	18	∞	∞	∞	∞	∞	∞	∞
4	3	2	5	10	9	14	19	18	21	∞	∞	∞	∞	∞	∞
5	3	1	4	3	8	11	10	13	20	19	22	∞	∞	∞	∞

An FPTAS for Knapsack

An FPTAS for Knapsack

If the profits of objects were small numbers, say, bounded by a polynomial in n, then the algorithm would be a regular polynomial time algorithm, since its running time would be bounded by a polynomial in $|I|$.

An FPTAS for Knapsack

If the profits of objects were small numbers, say, bounded by a polynomial in n, then the algorithm would be a regular polynomial time algorithm, since its running time would be bounded by a polynomial in $|I|$.

In FPTAS we will ignore a certain number of least significant bits of profits of objects (depending on ϵ), so that the modified profits can be viewed as numbers bounded by a polynomial in n and $1/\epsilon$.

An FPTAS for Knapsack

An FPTAS for Knapsack

1. Given $\epsilon > 0$, let
$$K = \frac{\epsilon P}{n}$$

An FPTAS for Knapsack

1. Given $\epsilon > 0$, let
$$K = \frac{\epsilon P}{n}$$

2. For each object a_i, define
$$\text{profit}'(a_i) = \lfloor \frac{\text{profit}(a_i)}{K} \rfloor$$

.

An FPTAS for Knapsack

1. Given $\epsilon > 0$, let
$$K = \frac{\epsilon P}{n}$$
2. For each object a_i, define
$$\text{profit}'(a_i) = \lfloor \frac{\text{profit}(a_i)}{K} \rfloor$$
.
3. With these as profits of objects, using the dynamic programming algorithm, find the most profitable set, say S'.

An FPTAS for Knapsack

1. Given $\epsilon > 0$, let
$$K = \frac{\epsilon P}{n}$$
2. For each object a_i, define
$$\text{profit}'(a_i) = \lfloor \frac{\text{profit}(a_i)}{K} \rfloor$$
.
3. With these as profits of objects, using the dynamic programming algorithm, find the most profitable set, say S'.
4. Output S'.

Analysis

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\text{profit}(A) \geq (1 - \epsilon) \cdot \text{OPT}.$$

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\text{profit}(A) \geq (1-\epsilon) \cdot \text{OPT}.$$

Proof:

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\text{profit}(A) \geq (1-\epsilon) \cdot \text{OPT}.$$

Proof:

- Let O denote the optimal set.

Analysis

Lemma
Let A denote the set output by the algorithm. Then

$$\text{profit}(A) \geq (1 - \epsilon) \cdot \text{OPT}.$$

Proof:

- Let O denote the optimal set.
- For any object a,

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\mathrm{profit}(A) \geq (1 - \epsilon) \cdot \mathrm{OPT}.$$

Proof:

- Let O denote the optimal set.
- For any object a,
  - because of rounding down, $K \cdot \mathrm{profit}'(a)$ can be smaller than $\mathrm{profit}(a)$,

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\text{profit}(A) \geq (1 - \epsilon) \cdot \text{OPT}.$$

Proof:

- Let O denote the optimal set.
- For any object a,
  - because of rounding down, $K \cdot \text{profit}'(a)$ can be smaller than $\text{profit}(a)$,
  - but by not more than K. Say, $\text{profit}(a) - K \cdot \text{profit}'(a) \leq K$

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\text{profit}(A) \geq (1-\epsilon) \cdot \text{OPT}.$$

Proof:

- Let O denote the optimal set.
- For any object a,
  - because of rounding down, $K \cdot \text{profit}'(a)$ can be smaller than $\text{profit}(a)$,
  - but by not more than K. Say, $\text{profit}(a) - K \cdot \text{profit}'(a) \leq K$
- Therefore,

$$\text{profit}(O) - K \cdot \text{profit}'(O) \leq nK$$

Analysis

Lemma
Let A denote the set output by the algorithm. Then

$$\text{profit}(A) \geq (1-\epsilon) \cdot \text{OPT}.$$

Proof:

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\text{profit}(A) \geq (1 - \epsilon) \cdot \text{OPT}.$$

Proof:

- The dynamic programming step must return a set at least as good as O under the new profits.

Analysis

Lemma

Let A denote the set output by the algorithm. Then

$$\text{profit}(A) \geq (1-\epsilon) \cdot \text{OPT}.$$

Proof:

- The dynamic programming step must return a set at least as good as O under the new profits.
- Therefore,

$$\begin{aligned}\text{profit}(S) &\geq K \cdot \text{profit}'(S) \geq K \cdot \text{profit}'(O) \\ &\geq \text{profit}(O) - nK = \text{OPT} - \epsilon P \geq (1-\epsilon) \cdot \text{OPT}\end{aligned}$$

Analysis

Analysis

By previous Lemma, the solution found is within $(1-\epsilon)$ factor of OPT. Since the running time of the algorithm is

$$O(n^2\lfloor\frac{P}{K}\rfloor) = O(n^2\lfloor\frac{n}{\epsilon}\rfloor)$$

which is polynomial in n and $1/\epsilon$, thus it is a FPTAS for knapsack.

Approximation Via LP

Set Cover

Set cover

Given a universe U of n elements, a collection of subsets of U, $\mathcal{S} = \{S_1, \ldots, S_k\}$, and a cost function $c : \mathcal{S} \to \mathbb{Q}^+$, find a minimum cost sub-collection of $\mathcal{S}$ that covers all elements of U.
The special case, in which all subsets are of unit cost, will be called the cardinality set cover problem.

The Set Cover in ILP

$$
\begin{aligned}
\text{minimize} \quad & \sum_{S \in \mathcal{S}} c(S) x_S & \\
\text{subject to} \quad & \sum_{S : e \in S} x_S \geq 1, & e \in U \\
& x_S \in \{0, 1\}, & S \in \mathcal{S}
\end{aligned}
$$

The Set Cover LP-Relaxation

$$\begin{aligned}
\text{minimize} \quad & \sum_{S \in \mathcal{S}} c(S) x_S & \\
\text{subject to} \quad & \sum_{S : e \in S} x_S \geq 1, & e \in U \\
& x_S \geq 0, & S \in \mathcal{S}
\end{aligned}$$

A Simple Rounding Algorithm

Algorithm

A Simple Rounding Algorithm

Algorithm

1. Find an optimal solution to the LP-relaxation.

A Simple Rounding Algorithm

Algorithm

1. Find an optimal solution to the LP-relaxation.
2. Pick all sets S for which $x_S \geq 1/f$ in this solution.

Analysis

Analysis

Theorem

This algorithm achieves an approximation factor of f for the set cover problem.

Analysis

Theorem
This algorithm achieves an approximation factor of f for the set cover problem.

Proof
Let $\mathcal{C}$ be the collection of picked sets. We first show that $\mathcal{C}$ is indeed a set cover.

Analysis

Theorem

This algorithm achieves an approximation factor of f for the set cover problem.

Proof

Let $\mathcal{C}$ be the collection of picked sets. We first show that $\mathcal{C}$ is indeed a set cover.

- Consider an element e. Since e is in at most f sets, one of these sets must be picked to the extent of at least $1/f$ in the fractional cover, due to the pigeonhole principle.

Analysis

Theorem

This algorithm achieves an approximation factor of f for the set cover problem.

Proof

Let $\mathcal{C}$ be the collection of picked sets. We first show that $\mathcal{C}$ is indeed a set cover.

- Consider an element e. Since e is in at most f sets, one of these sets must be picked to the extent of at least $1/f$ in the fractional cover, due to the pigeonhole principle.
- Thus, e is covered by $\mathcal{C}$, and hence $\mathcal{C}$ is a valid set cover.

Analysis

Analysis

- The rounding process increases x_S, for each set $S \in \mathcal{C}$, by a factor of at most f.

Analysis

- The rounding process increases x_S, for each set $S \in \mathcal{C}$, by a factor of at most f.
- Therefore, the cost of $\mathcal{C}$ is at most f times the cost of the fractional cover, thereby proving the desired approximation guarantee.

Exercise

[Als99] 15.6, 15.10, 15.12, 15.17, 15.19, 15.27