Algorithms and datastructures II

Lecture 11: Approximation algorithms

Jan Hubička

Department of Applied Mathematics
Charles University
Prague

Dec 14 2020

Decision problems

Definition

A (decision) problem if a function from $\{0,1\}^*$ (the set of all possible inputs) to $\{0,1\}$.

Definition (Reduction)

Given problems A and B, we say that A is (polynomial time) reducible to B (and write $A \longrightarrow B$) if there exists function $f : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$ it holds $A(x) = B(f(x))$ and f can be computed in polynomial time relative to $|x|$. Function f is also called (polynomial time) reduction.

Definition (P)

P is the class of all (decision) problems that can be solved by a polynomial time algorithm.

Definition (NP)

NP is the class of all (decision) problems L such that there exists some problem $K \in P$ and a polynomial g such that for every input x it holds that $L(x) = 1$ iff there exists $y \in \{0,1\}^*$ of length at most $g(|x|)$ such that $K(x, y) = 1$.

NP-completeness

Decision versus optimization problems

Definition (Decition problem)

A (decision) problem if a function from $\{0,1\}^*$ (the set of all possible inputs) to $\{0,1\}$.

Decision versus optimization problems

Definition (Decition problem)

A (decision) problem if a function from $\{0,1\}^*$ (the set of all possible inputs) to $\{0,1\}$.

Definition (Optimization problem)

Optimization problem has, for a given input x, set of acceptable solutions where every acceptable solution y has cost $c(y)$. We look for optimal solution with cost c^*.

Decision versus optimization problems

Definition (Decition problem)

A (decision) problem if a function from $\{0, 1\}^*$ (the set of all possible inputs) to $\{0, 1\}$.

Definition (Optimization problem)

Optimization problem has, for a given input x, set of acceptable solutions where every acceptable solution y has cost $c(y)$. We look for optimal solution with cost c^*.

Definition (α-approximation)

Given $\alpha > 1$, α-approximation is an acceptable solution to the optimization problem with cost c' satisfying

$$c' \leq \alpha c^*.$$

Relative error $(c' - c^*)/c^*$ is at most $\alpha - 1$.
Similarly we can study optimization problem maximizing the cost.

Vertex Cover Problem

Vertex cover of a graph is a set of vertices that includes at least one vertex of every edge of the graph.

Vertex Cover Problem

Vertex cover of a graph is a set of vertices that includes at least one vertex of every edge of the graph.

VertexCoverApprox (G)

Repeatedly take both endpoints of an edge and remove all edges containing them from graph.

Vertex Cover Problem

Vertex cover of a graph is a set of vertices that includes at least one vertex of every edge of the graph.

VertexCoverApprox (G)

Repeatedly take both endpoints of an edge and remove all edges containing them from graph.

Observation

Anwer of VertexCoverApprox is a vertex cover

Vertex Cover Problem

Vertex cover of a graph is a set of vertices that includes at least one vertex of every edge of the graph.

VertexCoverApprox (G)

Repeatedly take both endpoints of an edge and remove all edges containing them from graph.

Observation

Anwer of VertexCoverApprox is a vertex cover

Observation

Anwer of VertexCoverApprox is at most twice bigger than the optimal solution.

Vertex Cover Problem

Vertex cover of a graph is a set of vertices that includes at least one vertex of every edge of the graph.

VertexCoverApprox (G)

Repeatedly take both endpoints of an edge and remove all edges containing them from graph.

Observation

Anwer of VertexCoverApprox is a vertex cover

Observation

Anwer of VertexCoverApprox is at most twice bigger than the optimal solution.

Proof.

Every vertex cover contains at least one vertex from each edge considered by VertexCoverApprox. □

In 2005 Dinur and Safra proved that discovering 1.3606-approximation algorithm would imply $P = NP$.

Travelling salesman

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

This problem is NP-hard because the existence of hamiltonian cycle reduces to it.

Travelling salesman

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

This problem is NP-hard because the existence of hamiltonian cycle reduces to it. We show approximation algorithm if the distances satisfies the triangle inequality:

$$\ell(x, z) < \ell(x, y) + \ell(y, x) \text{ for all } x, y, z$$

TSPapprox (edge-labelled graph G)

1. Find minimum spanning tree T in G.

Travelling salesman

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

This problem is NP-hard because the existence of hamiltonian cycle reduces to it. We show approximation algorithm if the distances satisfies the triangle inequality:

$$\ell(x, z) < \ell(x, y) + \ell(y, x) \text{ for all } x, y, z$$

TSPapprox (edge-labelled graph G)

1. Find minimum spanning tree T in G.
2. Choose root r and execute DFS walk on T to produce a walk visiting all vertices

Travelling salesman

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

This problem is NP-hard because the existence of hamiltonian cycle reduces to it. We show approximation algorithm if the distances satisfies the triangle inequality:

$$\ell(x,z) < \ell(x,y) + \ell(y,x) \text{ for all } x, y, z$$

TSPapprox (edge-labelled graph G)

1. Find minimum spanning tree T in G.
2. Choose root r and execute DFS walk on T to produce a walk visiting all vertices
3. Use shortcuts instead of visiting every edge twice.

Travelling salesman

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

This problem is NP-hard because the existence of hamiltonian cycle reduces to it. We show approximation algorithm if the distances satisfies the triangle inequality:

$$\ell(x,z) < \ell(x,y) + \ell(y,x) \text{ for all } x, y, z$$

TSPapprox (edge-labelled graph G)

1. Find minimum spanning tree T in G.
2. Choose root r and execute DFS walk on T to produce a walk visiting all vertices
3. Use shortcuts instead of visiting every edge twice.

Theorem

Length of cycle given by TSPapprox not worse than twice the length of an optimal solution.

Proof.

Denote by $\ell(T)$ the length of T, by A the length of TSPapprox' solution and by O the length of the optimal answer.

Travelling salesman

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

This problem is NP-hard because the existence of hamiltonian cycle reduces to it. We show approximation algorithm if the distances satisfies the triangle inequality:

$$\ell(x,z) < \ell(x,y) + \ell(y,x) \text{ for all } x,y,z$$

TSPapprox (edge-labelled graph G)

1. Find minimum spanning tree T in G.
2. Choose root r and execute DFS walk on T to produce a walk visiting all vertices
3. Use shortcuts instead of visiting every edge twice.

Theorem

Length of cycle given by TSPapprox not worse than twice the length of an optimal solution.

Proof.

Denote by $\ell(T)$ the length of T, by A the length of TSPapprox' solution and by O the length of the optimal answer.

$$A < 2\ell(T)$$

Travelling salesman

Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?

This problem is NP-hard because the existence of hamiltonian cycle reduces to it. We show approximation algorithm if the distances satisfies the triangle inequality:

$$\ell(x, z) < \ell(x, y) + \ell(y, x) \text{ for all } x, y, z$$

TSPapprox (edge-labelled graph G)

1. Find minimum spanning tree T in G.
2. Choose root r and execute DFS walk on T to produce a walk visiting all vertices
3. Use shortcuts instead of visiting every edge twice.

Theorem

Length of cycle given by TSPapprox not worse than twice the length of an optimal solution.

Proof.

Denote by $\ell(T)$ the length of T, by A the length of TSPapprox' solution and by O the length of the optimal answer.

$$A < 2\ell(T) \leq 2O.$$

Travelling salesman

Theorem

If exists an t-approximation algorithm for TSP (without the assumption of triangle inequality), for some $t \geq 1$, there then $P = NP$.

Travelling salesman

Theorem

If exists an t-approximation algorithm for TSP (without the assumption of triangle inequality), for some $t \geq 1$, there then $P = NP$.

Proof.

We show that we can use this algorithm to decide on existence of the hamiltonian cycle.

1. Given graph $G = (V, E)$ complete it to G'. Edges of G has length 1 and non-edges some length c.

Travelling salesman

Theorem

If exists an t-approximation algorithm for TSP (without the assumption of triangle inequality), for some $t \geq 1$, there then $P = NP$.

Proof.

We show that we can use this algorithm to decide on existence of the hamiltonian cycle.

1. Given graph $G = (V, E)$ complete it to G'. Edges of G has length 1 and non-edges some length c.
2. If G has hamiltonian cycle then TSP solution for G' has length $n = |E|$.

Travelling salesman

Theorem

If exists an t-approximation algorithm for TSP (without the assumption of triangle inequality), for some $t \geq 1$, there then $P = NP$.

Proof.

We show that we can use this algorithm to decide on existence of the hamiltonian cycle.

1. Given graph $G = (V, E)$ complete it to G'. Edges of G has length 1 and non-edges some length c.
2. If G has hamiltonian cycle then TSP solution for G' has length $n = |E|$.
3. If G has no hamiltonian cycle then the TSP solution for G' has length at least $n - 1 + c$.

Travelling salesman

Theorem

If exists an t-approximation algorithm for TSP (without the assumption of triangle inequality), for some $t \geq 1$, there then $P = NP$.

Proof.

We show that we can use this algorithm to decide on existence of the hamiltonian cycle.

1. Given graph $G = (V, E)$ complete it to G'. Edges of G has length 1 and non-edges some length c.
2. If G has hamiltonian cycle then TSP solution for G' has length $n = |E|$.
3. If G has no hamiltonian cycle then the TSP solution for G' has length at least $n - 1 + c$.
4. We want $tn < n - 1 + c$, so we can chose $c > (t - 1)n + 1$.

□

Recall: Knapsack problem

Knapsack problem

Given set of n objects with weights $w_1, \dots, w_n$, costs $c_1, \dots, c_n$ and maximum weight W your knapsack can carry. Find subset $P \subseteq \{1, 2, \dots, n\}$ such that $w(P) = \sum_{i \in P} w_i$ is at most W and the cost $c(P) = \sum_{i \in P} c_i$ is maximum possible.

We can use dynamic programming to solve the problem in polynomial time in $C = \sum c_i$.

1. Denote by $A_k(c)$ the minimum of weights of subsets $P \subseteq \{1, 2, \dots, k\}$ satisfying $c(P) = c$.
2. Proceed by induction:
   1. $A_0(0) = 0$, $A_0(1) = A_0(2) = \dots = A_0(C) = \infty$.
   2. Given A_{k-1} compute
   $$A_k(c) = \min(A_{k-1}(c), A_{k-1}(c - c_k) + w_k)$$
3. Once A_n is determined we know for every possible cost the subset P of that cost minimizing the weight. It remains to find maximal c such that $A_n(c) \leq W$
4. To determine the set P one can remember how the values $A_k(c)$ was determined.

(This is pseudo-polynomial algorithm) running in time $O(nC)$.

KnapsackApprox ($w_1, \dots, w_n, c_1, \dots, c_n, \epsilon$)

1. Remove from input all items heavier than W
2. Compute $c_{\max} = \max_i c_i$ and choose $M = \lfloor n/\epsilon \rfloor$.
3. For $i = 1, \dots, n$ put $\hat{c} \leftarrow \lfloor c_i \cdot M / c_{\max} \rfloor$.
4. Apply dynamic programming to solve knapsack with costs $\hat{c}_1, \dots \hat{c}_n$.
5. Return solution with same items as chosen by the approximate solution.

KnapsackApprox $(w_1, \ldots, w_n, c_1, \ldots, c_n, \epsilon)$

1. Remove from input all items heavier than W
2. Compute $c_{\max} = \max_i c_i$ and choose $M = \lfloor n/\epsilon \rfloor$.
3. For $i = 1, \ldots, n$ put $\hat{c} \leftarrow \lfloor c_i \cdot M/c_{\max} \rfloor$.
4. Apply dynamic programming to solve knapsack with costs $\hat{c}_1, \ldots \hat{c}_n$.
5. Return solution with same items as chosen by the approximate solution.

$\hat{C} \leq nM = O(n^2/\epsilon)$ and thus runtime is $o(n\hat{C}) = O(n^3/\epsilon)$.

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

KnapsackApprox $(w_1, \ldots, w_n, c_1, \ldots, c_n, \epsilon)$

1. Remove from input all items heavier than W
2. Compute $c_{\max} = \max_i c_i$ and choose $M = \lfloor n/\epsilon \rfloor$.
3. For $i = 1, \ldots, n$ put $\hat{c} \leftarrow \lfloor c_i \cdot M / c_{\max} \rfloor$.
4. Apply dynamic programming to solve knapsack with costs $\hat{c}_1, \ldots \hat{c}_n$.
5. Return solution with same items as chosen by the approximate solution.

$\hat{C} \leq nM = O(n^2/\epsilon)$ and thus runtime is $o(n\hat{C}) = O(n^3/\epsilon)$.

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) \quad =$$

KnapsackApprox $(w_1, \dots, w_n, c_1, \dots, c_n, \epsilon)$

1. Remove from input all items heavier than W
2. Compute $c_{\max} = \max_i c_i$ and choose $M = \lfloor n/\epsilon \rfloor$.
3. For $i = 1, \dots, n$ put $\hat{c} \leftarrow \lfloor c_i \cdot M / c_{\max} \rfloor$.
4. Apply dynamic programming to solve knapsack with costs $\hat{c}_1, \dots \hat{c}_n$.
5. Return solution with same items as chosen by the approximate solution.

$\hat{C} \leq nM = O(n^2/\epsilon)$ and thus runtime is $o(n\hat{C}) = O(n^3/\epsilon)$.

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) \quad = \quad \sum_{i \in P} \hat{c}_i =$$

KnapsackApprox $(w_1, \ldots, w_n, c_1, \ldots, c_n, \epsilon)$

1. Remove from input all items heavier than W
2. Compute $c_{\max} = \max_i c_i$ and choose $M = \lfloor n/\epsilon \rfloor$.
3. For $i = 1, \ldots, n$ put $\hat{c} \leftarrow \lfloor c_i \cdot M/c_{\max} \rfloor$.
4. Apply dynamic programming to solve knapsack with costs $\hat{c}_1, \ldots \hat{c}_n$.
5. Return solution with same items as chosen by the approximate solution.

$\hat{C} \leq nM = O(n^2/\epsilon)$ and thus runtime is $o(n\hat{C}) = O(n^3/\epsilon)$.

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq$$

KnapsackApprox $(w_1, \dots, w_n, c_1, \dots, c_n, \epsilon)$

1. Remove from input all items heavier than W
2. Compute $c_{\max} = \max_i c_i$ and choose $M = \lfloor n/\epsilon \rfloor$.
3. For $i = 1, \dots, n$ put $\hat{c} \leftarrow \lfloor c_i \cdot M / c_{\max} \rfloor$.
4. Apply dynamic programming to solve knapsack with costs $\hat{c}_1, \dots \hat{c}_n$.
5. Return solution with same items as chosen by the approximate solution.

$\hat{C} \leq nM = O(n^2/\epsilon)$ and thus runtime is $o(n\hat{C}) = O(n^3/\epsilon)$.

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i \in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq$$

KnapsackApprox $(w_1, \ldots, w_n, c_1, \ldots, c_n, \epsilon)$

1. Remove from input all items heavier than W
2. Compute $c_{\max} = \max_i c_i$ and choose $M = \lfloor n/\epsilon \rfloor$.
3. For $i = 1, \ldots, n$ put $\hat{c} \leftarrow \lfloor c_i \cdot M / c_{\max} \rfloor$.
4. Apply dynamic programming to solve knapsack with costs $\hat{c}_1, \ldots \hat{c}_n$.
5. Return solution with same items as chosen by the approximate solution.

$\hat{C} \leq nM = O(n^2/\epsilon)$ and thus runtime is $o(n\hat{C}) = O(n^3/\epsilon)$.

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) \quad = \quad \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i \in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i \in P} c_i \cdot \frac{M}{c_{\max}} \right) - n =$$

KnapsackApprox $(w_1, \ldots, w_n, c_1, \ldots, c_n, \epsilon)$

1. Remove from input all items heavier than W
2. Compute $c_{\max} = \max_i c_i$ and choose $M = \lfloor n/\epsilon \rfloor$.
3. For $i = 1, \ldots, n$ put $\hat{c} \leftarrow \lfloor c_i \cdot M / c_{\max} \rfloor$.
4. Apply dynamic programming to solve knapsack with costs $\hat{c}_1, \ldots \hat{c}_n$.
5. Return solution with same items as chosen by the approximate solution.

$\hat{C} \leq nM = O(n^2/\epsilon)$ and thus runtime is $o(n\hat{C}) = O(n^3/\epsilon)$.

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i \in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i \in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

□

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i\in P} \hat{c}_i = \sum_{i\in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i\in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i\in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

$$c(Q) =$$

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i \in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i \in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

$$c(Q) = \sum_{i \in Q} c_i \geq$$

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i \in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i \in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

$$c(Q) = \sum_{i \in Q} c_i \geq \sum_{i \in Q} \hat{c}_i \cdot \frac{c_{\max}}{M} =$$

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i\in P} \hat{c}_i = \sum_{i\in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i\in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i\in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

$$c(Q) = \sum_{i\in Q} c_i \geq \sum_{i\in Q} \hat{c}_i \cdot \frac{c_{\max}}{M} = \left(\sum_{i\in Q} \hat{c}_i \right) \cdot \frac{c_{\max}}{M} =$$

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i \in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i \in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

$$c(Q) = \sum_{i \in Q} c_i \geq \sum_{i \in Q} \hat{c}_i \cdot \frac{c_{\max}}{M} = \left(\sum_{i \in Q} \hat{c}_i \right) \cdot \frac{c_{\max}}{M} = \hat{c}(Q) \cdot \frac{c_{\max}}{M} \geq$$

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i \in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i \in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

$$c(Q) = \sum_{i \in Q} c_i \geq \sum_{i \in Q} \hat{c}_i \cdot \frac{c_{\max}}{M} = \left(\sum_{i \in Q} \hat{c}_i \right) \cdot \frac{c_{\max}}{M} = \hat{c}(Q) \cdot \frac{c_{\max}}{M} \geq \hat{c}(P) \cdot \frac{c_{\max}}{M}.$$

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i \in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i \in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

$$c(Q) = \sum_{i \in Q} c_i \geq \sum_{i \in Q} \hat{c}_i \cdot \frac{c_{\max}}{M} = \left(\sum_{i \in Q} \hat{c}_i \right) \cdot \frac{c_{\max}}{M} = \hat{c}(Q) \cdot \frac{c_{\max}}{M} \geq \hat{c}(P) \cdot \frac{c_{\max}}{M}.$$

$$c(Q) \geq \left(c(P) \cdot \frac{M}{c_{\max}} - n \right) \cdot \frac{c_{\max}}{M} \geq$$

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i \in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i \in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

$$c(Q) = \sum_{i \in Q} c_i \geq \sum_{i \in Q} \hat{c}_i \cdot \frac{c_{\max}}{M} = \left(\sum_{i \in Q} \hat{c}_i \right) \cdot \frac{c_{\max}}{M} = \hat{c}(Q) \cdot \frac{c_{\max}}{M} \geq \hat{c}(P) \cdot \frac{c_{\max}}{M}.$$

$$c(Q) \geq \left(c(P) \cdot \frac{M}{c_{\max}} - n \right) \cdot \frac{c_{\max}}{M} \geq c(P) - \frac{n \cdot c_{\max}}{n/\epsilon} \geq$$

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i\in P} \hat{c}_i = \sum_{i\in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i\in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i\in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

$$c(Q) = \sum_{i\in Q} c_i \geq \sum_{i\in Q} \hat{c}_i \cdot \frac{c_{\max}}{M} = \left(\sum_{i\in Q} \hat{c}_i \right) \cdot \frac{c_{\max}}{M} = \hat{c}(Q) \cdot \frac{c_{\max}}{M} \geq \hat{c}(P) \cdot \frac{c_{\max}}{M}.$$

$$c(Q) \geq \left(c(P) \cdot \frac{M}{c_{\max}} - n \right) \cdot \frac{c_{\max}}{M} \geq c(P) - \frac{n \cdot c_{\max}}{n/\epsilon} \geq c(P) - \epsilon c_{\max} \geq$$

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i\in P} \hat{c}_i = \sum_{i\in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i\in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i\in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

$$c(Q) = \sum_{i\in Q} c_i \geq \sum_{i\in Q} \hat{c}_i \cdot \frac{c_{\max}}{M} = \left(\sum_{i\in Q} \hat{c}_i \right) \cdot \frac{c_{\max}}{M} = \hat{c}(Q) \cdot \frac{c_{\max}}{M} \geq \hat{c}(P) \cdot \frac{c_{\max}}{M}.$$

$$c(Q) \geq \left(c(P) \cdot \frac{M}{c_{\max}} - n \right) \cdot \frac{c_{\max}}{M} \geq c(P) - \frac{n \cdot c_{\max}}{n/\epsilon} \geq c(P) - \epsilon c_{\max} \geq c(P) - \epsilon c(P) =$$

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i \in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq \left(\sum_{i \in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

$$c(Q) = \sum_{i \in Q} c_i \geq \sum_{i \in Q} \hat{c}_i \cdot \frac{c_{\max}}{M} = \left(\sum_{i \in Q} \hat{c}_i \right) \cdot \frac{c_{\max}}{M} = \hat{c}(Q) \cdot \frac{c_{\max}}{M} \geq \hat{c}(P) \cdot \frac{c_{\max}}{M}.$$

$$c(Q) \geq \left(c(P) \cdot \frac{M}{c_{\max}} - n \right) \cdot \frac{c_{\max}}{M} \geq c(P) - \frac{n \cdot c_{\max}}{n/\epsilon} \geq c(P) - \epsilon c_{\max} \geq c(P) - \epsilon c(P) = (1 - \epsilon) \cdot c(P).$$

□

Theorem

Solution by KnapsackApprox has relative error at most ϵ.

Proof.

Let P be optimal solution, $c(P)$ its cost, Q KnapsackApprox' solution and cost $\hat{c}(Q)$. We need to determine $c(Q)$.

$$\hat{c}(P) = \sum_{i\in P} \hat{c}_i = \sum_{i\in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i\in P}\left(c_i \cdot \frac{M}{c_{\max}} - 1\right) \geq \left(\sum_{i\in P} c_i \cdot \frac{M}{c_{\max}}\right) - n = c(P)\cdot\frac{M}{c_{\max}} - n.$$

$$c(Q) = \sum_{i\in Q} c_i \geq \sum_{i\in Q} \hat{c}_i \cdot \frac{c_{\max}}{M} = \left(\sum_{i\in Q}\hat{c}_i\right)\cdot\frac{c_{\max}}{M} = \hat{c}(Q)\cdot\frac{c_{\max}}{M} \geq \hat{c}(P)\cdot\frac{c_{\max}}{M}.$$

$$c(Q) \geq \left(c(P)\cdot\frac{M}{c_{\max}} - n\right)\cdot\frac{c_{\max}}{M} \geq c(P) - \frac{n\cdot c_{\max}}{n/\epsilon} \geq c(P) - \epsilon c_{\max} \geq c(P) - \epsilon c(P) = (1-\epsilon)\cdot c(P).$$

□

Definition

Algorithm which for every $\epsilon > 0$ finds in a polynomial time $(1-\epsilon)$-approximation is called polynomial-time approximation scheme (PTAS).
If the time complexity is also polynomial in $1/\epsilon$ it is called full polynomial-time approximation scheme (FPTAS).