アルゴリズム論
Theory of Algorithms

第13回講義
近似アルゴリズム

アルゴリズム論
Theory of Algorithms

Lecture #13
Approximation Algorithm

近似アルゴリズムの必要性

実際の問題の多くは多項式時間では解けそうもない．
(1) 重みつきグラフが与えられたとき，すべての重みが正なら，任意の2点間の最短経路は多項式時間で求まる．
(2) 同じ状況で，すべての頂点を訪問する最短の閉路を求める問題はNP完全なので，多項式時間では解けそうもない．（巡回セールスパーソン問題）
(3) 重みのないグラフにおいても，すべての頂点をちょうど1度だけ通る閉路が存在するかどうかを判定する問題でもNP完全である．（ハミルトン閉路問題）
(4) 重みつきグラフにおいて，長さが最小であるような単純な経路（同じ頂点を1度しか通らない経路）を求める問題は，負の重みを許さないなら多項式時間で解けるが，負の重みを許すとNP困難である．
(5) グラフにおいて最長の単純経路を求める問題は，たとえ重みのないグラフでもNP困難である．（最長路問題）

Necessity of approximation algorithm

Most of practical problems seem to be intractable.
(1) Given a weighted graph, if every weight is positive, a shortest path between any two points can be solved in polynomial time.
(2) Since the problem of finding a shortest tour visiting all the vertices in the same situation is NP-complete, it seems to be intractable (Travelling Salesperson Problem).
(3) Even in an unweighted graph, the problem of determining if there is a closed loop visiting every vertex exactly once is also NP-complete (Hamiltonian cycle problem).
(4) The problem of finding a shortest simple cycle (a path visiting every vertex exactly once) in a weighted graph can be solved in polynomial time if negative weight is not allowed, but it is NP-hard if negative weight is allowed.
(5) The problem of finding a longest simple path in a graph is NP-hard even if the graph is not weighted (longest path problem).

近似アルゴリズムの必要性

実際にはNP困難の問題が多いので，最適解を妥当な時間内に求めることは困難．➔近似解での代用
このとき，単に近似解を求めるだけでなく，近似性能を保証することが重要．

近似性能，近似率の定義

ここでは，目的関数の値を最小化する最適化問題を考える．
最適化問題Pに対して，
S: 実行可能解　評価値 val(S)
Opt: 最適解　評価値 val(Opt)

近似率（近似性能）δ= val(S) / val(Opt).
（最大化問題の場合には val(Opt) / val(S)と定義
δ-近似アルゴリズム=近似率δ以下の解を求める近似アルゴリズムのこと
ただし，δは入力のサイズに無関係であること．

Necessity of approximation algorithm

Many practical problems are NP-hard and so it seems hard to solve them. ➔substitution by approximate solution
Then, it is important not only to find an approximate solution but also to guarantee the performance ratio of approximation.

Performance ratio of approximation and definition of approximation ratio

Consider an optimization problem of minimizing an objective function. For an optimization problem P,
S: a feasible solution　value of S : val(S)
Opt: optimal solution　its value : val(Opt)

Performance ratio (approximation ratio)δ= val(S) / val(Opt).
(it is defined as val(Opt) / val(S) for maximization problem)
δ-approximation algorithm=approximation algorithm with performance ration at most δ
Here, δmust be independent of an input size.

相対誤差ε= |val(Opt) – val(S)| / val(Opt)

相対誤差がε以内の解を求める近似アルゴリズム
= (1+ ε)-近似アルゴリズム
εも入力のサイズに無関係でなければならない．

理論的な興味：
最適化問題に対して多項式時間の(1+ ε)-近似アルゴリズム
を見つけること，あるいはそれが不可能であることを示すこと．

多項式時間近似方式(PTAS)
最適化問題に対する (1+ ε)-近似アルゴリズムで，計算時間
が入力サイズの多項式で抑えられるもの．

完全多項式時間近似方式(FPTAS)
最適化問題に対する (1+ ε)-近似アルゴリズムで，計算時間
が入力サイズと1/εの多項式で抑えられるもの．

もちろん，どんな最適化問題に対しても多項式時間近似方式が
存在するわけではない．定数近似の多項式アルゴリズムさえ
存在しない例も多い

Relative errorε= |val(Opt) – val(S)| / val(Opt)

Approximation algorithm of finding an approximation solution
with relative error at most ε= (1+ ε)-approximation algorithm
εmust be independent of an input size, too.

Theoretical interest:
To find a (1+ ε)-approximation algorithm for an optimization
problem or to show that it is impossible.

Polynomial-Time Approximation Scheme (PTAS)
a (1+ ε)-approximation algorithm for an optimization problem
with computation time bounded by a polynomial in input size

Fully Polynomial-Time Approximation Scheme (FPTAS)
a (1+ ε)-approximation algorithm for an optimization problem
with computation time bounded by a polynomial in input size and 1/ε

Of course, polynomial-time approximation scheme does not always
exist for any optimization problem. In some case there is no
approximation algorithm with constant approximation ratio.

問題P33:（ナップサック問題）
n個の荷物o_i(i=1, ... , n)に対する重さw_iと価値v_i，ナップサックの
制限重量Cが与えられたとき，荷物の合計の重さがCを超えない
ような荷物の詰め込み方の中で価値が最大となるものを求めよ．

入力を$I = \{w_1, \dots, w_n; v_1, \dots, v_n; C\}$とする．解は$\{1,2,\dots,n\}$の
部分集合Sで表現できる．最適解は，

容量制約 $\sum_{i\in S} w_i \leqq C$

を満たすSの中で

価値の総和 $\sum_{i\in S} v_i$

を最大にするものである．

仮定：どの荷物についても，その重さはCを超えないものとする．
Cを超える荷物は決して選ばれることがないからである．

10回目の講義でも同じ問題を扱ったが，ここでは重さが整数と
は限らない一般の場合を考える．➔動的計画法は使えない．

Problem P33:（Knapsack problem）
Given n objects o_i(i=1, ... , n) with their weight w_i and value v_i, with
weight limit C, choose objects to maximize the sum of values
so that the total weight is at most C.

Let input be $I = \{w_1, \dots, w_n; v_1, \dots, v_n; C\}$. A solution can be
represented as a subset S of $\{1,2,\dots,n\}$. An optimal solution is a
subset S that satisfies

weight constraint $\sum_{i\in S} w_i \leqq C$

and maximizes

total sum of values $\sum_{i\in S} v_i$

Assumption：Weight of any object does not exceed C. Any such
object is never selected even if there is one.

The same problem was dealt with in the 10th lecture, but consider
a general case in which weight is not integer.
➔Dynamic Programming cannot be used.

アルゴリズムP33-A0:（貪欲法）
(1) 単位重さあたりの価値v_i / w_iの降順にソートする．
$v_1 / w_1 \geqq v_2 / w_2 \geqq \cdots \geqq v_n / w_n$
(2) 上記のソート順に従って荷物をナップサックに入れていく．
容量制約を満たさなくなれば，最後の荷物を取り除いて終り．

このアルゴリズムは第5回の講義で説明済み．

演習問題E14-1：貪欲法では非常に悪い解しか得られない例題を
作り，その解を最適解と比較せよ．

荷物を分割可能な一般化ナップサック問題に対しては上記の
方法で最後の荷物を分割すれば最適解が得られる．
実際にも，上記のアルゴリズムで良い解が得られることは多い．
しかし，荷物の分割を許さない場合には最適解が得られる保証は
ない．
では，近似アルゴリズムと考えたとき，近似率はどの程度か？

Algorithm P33-A0: (Greedy algorithm)
(1) Sort objects in decreasing order of v_i / w_i, value per unit weight.
$v_1 / w_1 \geqq v_2 / w_2 \geqq \cdots \geqq v_n / w_n$
(2) Put objects into knapsack in the sorted order. When the weight
constraint is not satisfied, stop after removing the last object.

This algorithm was explained in Lecture 5.

Exercise E14-1：Create an example in which the greedy algorithm
finds only very bad solutions and compare those solutions with an
optimal solution.

In a generalized knapsack problem in which any object can be
divided, the above algorithm finds an optimal solution by dividing
the last object. In practice, the algorithm finds good solutions in
many cases. But there is no guarantee for optimal solution if object
cannot be divided.

Evaluate its performance ratio as an approximation algorithm

観察：アルゴリズムP33-A0によって得られる解の近似率はどんな定数でも抑えられない．

証明：
任意の定数C≧3に対して，次のようなナップサック問題を考える：
荷物は2つ（$w_1 = 1, v_1 = 2, w_2 = C, v_2 = C$）
容量制約はC
➔単位重さ当たりの価値でソートすると1,2の順
アルゴリズムでは荷物1だけの解を得る
よって，$val(S) = v_1 = 2$
一方，荷物2だけを選ぶことができて，そのときの価値は
$val(Opt) = v_2 = C$.
両者の比（最大化問題に対する比）を取ると，
$val(Opt) / val(S) = C/2$
となる．定数Cは任意だから，近似率は任意の値に設定できる．
証明終

Observation: The approximation ratio of a solution obtained by the algorithm P33-A0 cannot be bounded by any constant.

Proof:
Consider the following knapsack problem for any constant C≧3:
only two objects（$w_1 = 1, v_1 = 2, w_2 = C, v_2 = C$）
weight limit is C
➔sorted order by value per unit weight: 1,2
In the algorithm only object 1 is included in the solution.
Thus, $val(S) = v_1 = 2$.
On the other hand, if we choose object 2, its value is
$val(Opt) = v_2 = C$.
Taking their ratio between them (for maximization problem),
$val(Opt) / val(S) = C/2$.
Constant C is arbitrary. So, the approximation ratio is arbitrary.
Q.E.D.

近似率の改善

貪欲法では単位重さ当たりの価値だけに注目した．
単位重さ当たりではなく，価値そのものが最大の荷物にも注目．

アルゴリズムP33-A1: (貪欲法の改良)
(1) 単位重さあたりの価値v_i / w_iの降順にソートする．
$v_1 / w_1 \geqq v_2 / w_2 \geqq \cdots \geqq v_n / w_n$
(2) 上記のソート順に従って荷物をナップサックに入れていく．
容量制約を満たさなくなれば，最後の荷物を取り除く．
(3) 上で得られたを解S_1とし，価値が最大の品物の価値をv_kとする
$val(S_1) < v_k$ ならば{k}を解とする；そうでなければS_1を解とする．

今度のアルゴリズムの近似率は2であることが示せる：
アルゴリズムの出力をSとすると，
$val(S) = \max\{val(S_1), v_k\}$.
Sが全体集合{1,2, ... , n}なら，明らかに最適解である．
そこで，Sに含まれない荷物があると仮定する：最小の番号をiとする

Improving the performance ratio

Greedy algorithm considered only values per unit weight.
Consider not only them but also the object of the largest value.

Algorithm P33-A1: (Improvement of Greey Algorithm)
(1) Sort objects in decreasing order of values per unit weight v_i / w_i.
$v_1 / w_1 \geqq v_2 / w_2 \geqq \cdots \geqq v_n / w_n$
(2) Put objects into knapsack in the sorted order.
When exceeding the weight limit, delete the last object.
(3) Let the solution obtained be S_1 and let v_k be the largest value.
if $val(S_1) < v_k$ then let {k} be the solution else let S_1 be solution.

We can show that this algorithm is a 2-approximation algorithm:
Let S be the output of the algorithm. Then,
$val(S) = \max\{val(S_1), v_k\}$.
If S is the whole set {1,2, ... , n}, it is obviously optimal.
So, assume that at least one object is not in S. Let i be such an object of the least number

荷物i: アルゴリズムの解に含まれない番号最小の荷物
荷物の分割を許す場合には貪欲法で最適解が得られるから，
$val(Opt) < v_1 + v_2 + \cdots + v_i = val(S_1) + v_i$
が成り立つ．また，仮定より，$v_i \leqq v_k$.
よって，
$v_1 + v_2 + \cdots + v_i \leqq val(S_1) + v_i \leqq val(S_1) + v_k \leqq 2\max\{val(S_1), v_k\}$
$= 2val(S)$.
したがって，
$val(Opt) < 2val(S)$,
つまり，近似率は2である．　証明終

近似率を更に改善することは可能か？
(1+ε)-近似アルゴリズム？

Object i: the object of the least number that is not contained in the solution given by the algorithm.
Since an optimal solution is obtained by Greedy Algorithm if objects can be partitioned, we have
$val(Opt) < v_1 + v_2 + \cdots + v_i = val(S_1) + v_i$
Also, by the assumption $v_i \leqq v_k$.
Thus, we have
$v_1 + v_2 + \cdots + v_i \leqq val(S_1) + v_i \leqq val(S_1) + v_k \leqq 2\max\{val(S_1), v_k\}$
$= 2val(S)$.
and hence
$val(Opt) < 2val(S)$,
that is, the performance ratio is 2.　Q.E.D

Can we further improve the performance ratio?
(1+ε)-approximation algorithm?

アルゴリズムP33-A2: 多項式時間近似方式

ε:任意の小さな定数
$\varepsilon \geqq 1/k$を満たす最小の正整数をkとする.
要素数がk以下のすべての部分集合を生成し，それぞれの部分集合Tを次のように成長させる:
・部分集合Tの要素数は高々k,
・部分集合Tの総重量が容量制約を超過していればTを無視.
・超過していなければ，Tに含まれない荷物を単位重さ当たりの価値が大きい順に容量制約を満たす限りTに加える.
上記のようにして得られた解の内で最も良いものを出力する.

要素数がk以下の部分集合は，$O(n^k)$通りある
　kは入力サイズに無関係な定数なので，$O(n^k)$は多項式.
　各部分集合に対する貪欲拡張操作は$O(n)$時間.
　よって，全体の計算時間は$O(n^{k+1})$: 多項式時間.
次に，このアルゴリズムが$(1+\varepsilon)$-近似アルゴリズムであることを示そう.

Opt: 最適解とする.
　$|Opt| \leqq k$→アルゴリズムで必ず見つかっている.
　　∵アルゴリズムでは要素数k以下の部分集合を全て調べる.
そこで，以下では$|Opt|>k$と仮定する.

Algorithm P33-A2: Polynomial-Time Approximation Scheme

ε: arbitrary small constant
let k be a positive integer such that $\varepsilon \geqq 1/k$
Generate all the subsets of size at most k, and extend each such subset T as follows:
・cardinality of T is at most k.
・Neglect the subset T if its total weight exceeds the weight limit.
・If it is within the limit, put the objects not in T into T in the decreasing order of their values per unit weight.
Output the best solution among those obtained above.

There are $O(n^k)$ different subsets of size at most k.
　Since k is a constant independent of input size, $O(n^k)$ is a polynomial.
　Greedy extension operation for each subset is done in $O(n)$ time.
　Thus, the total time is $O(n^{k+1})$: polynomial.
Next, we shall show this is a $(1+\varepsilon)$-approximation algorithm.

Opt: optimal solution
　$|Opt| \leqq k$→it must have been found in the algorithm.
　　∵ Algorithm examines all subsets of size at most k.
So, we assume $|Opt|>k$ in the following.

$|Opt| \geqq k+1$と仮定する:
　$Opt = \{i_1, i_2, \ldots, i_r\}$, $r \geqq k+1$, $v(i_1) \geqq v(i_2) \geqq \cdots \geqq v(i_r)$ と仮定

　最初のk個の荷物からなる集合をXとし，Optの残りの荷物を単位重さ当たりの価値の降順にソートする.
　$Opt = \{i_1, i_2, \ldots, i_k, i_{k+1}, \ldots, i_{m-1}, i_m, \ldots, i_r\}$
　　| 集合X | この部分は単位重さ当たりの価値の降順

　このとき，集合X以外のOptの要素i_jについては,
　　$v(i_j) \leqq val(Opt)/(k+1)$, $j=k+1, \ldots, r$
　が成り立つ.

　集合Xから始めて，Xに属さない荷物を，容量制約を満たす限り，単位重さ当たりの価値の降順に加えていき，集合Sを得る.
　このとき，S=Optなら，最適解が見つかっているから問題ない.
$S \neq Opt$→Opt-Sの荷物の中で単位重さあたりの価値が最大のものをi_mとする.
つまり,
　　$v(i_m)/w(i_m) \geqq v(i_q)/w(i_q)$, $q=m+1, \ldots, r$　　(A)
　$S = \{i_1, i_2, \ldots, i_k, i_{k+1}, \ldots, i_{m-1}, j_1, \ldots, j_s\}$
　　| ここまでOptと同じ　| 後は貪欲に拡張した部分

Assume $|Opt| \geqq k+1$:
　Suppose that $Opt = \{i_1, i_2, \ldots, i_r\}$, $r \geqq k+1$, $v(i_1) \geqq v(i_2) \geqq \cdots \geqq v(i_r)$

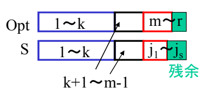
　Let X be a set of the first k objects, and sort the remaining objects of OPT in the decreasing order of values per unit weight.
　$Opt = \{i_1, i_2, \ldots, i_k, i_{k+1}, \ldots, i_{m-1}, i_m, \ldots, i_r\}$
　　| set X | this part is in the decreasing order of values per unit weight

　Then, for each element i_j of Opt not in X we have
　　$v(i_j) \leqq val(Opt)/(k+1)$, $j=k+1, \ldots, r$.

　Starting from a set X, we put objects not in X within the weight limit in the decreasing order of values per unit weight. Let the resulting set be S.
　Then, if S=Opt, there is no problem since we have found an optimal solution.
　Let i_m be an object of the largest value per unit weight in $S \neq Opt$→Opt-S.
　That is,
　　$v(i_m)/w(i_m) \geqq v(i_q)/w(i_q)$, $q=m+1, \ldots, r$　　(A)
　$S = \{i_1, i_2, \ldots, i_k, i_{k+1}, \ldots, i_{m-1}, j_1, \ldots, j_s\}$
　　| same as Opt thus far　| the remaining is the part extended by Greedy Algorithm

このとき，貪欲法の性質より，Sで加えた荷物は単位重さ当たりの価値では荷物i_m以上だから(そうでなければ，i_mを加えたはず)
　$v(j_t)/w(j_t) \geqq v(i_m)/w(i_m)$, $t=1, \ldots, s$
が成り立つ.

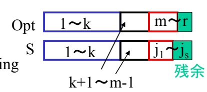
S*をアルゴリズムで求められる近似解とすると,
　$val(S^*) \geqq val(S) = \sum_{j=1\sim k} v(i_j) + \sum_{j=k+1\sim m-1} v(i_j) + \sum_{t=1\sim s} v(j_t)$
ここで，Optでのi_mまでの部分の残余容量C’と，Sでの残余容量C”は
　$C' = C - \sum_{j=1\sim k} v(i_j) - \sum_{j=k+1\sim m-1} v(i_j)$,
　$C'' = C' - \sum_{t=1\sim s} v(j_t)$



残余容量C’をすべて$v(i_m)/w(i_m)$の価値で詰めた方が価値は高くなるから,
　$val(Opt) \leqq \sum_{j=1\sim k} v(i_j) + \sum_{j=k+1\sim m-1} v(i_j) + C' v(i_m)/w(i_m)$.
i_mはSに含まれず，最後にSに含める余裕もないから,
　$C'' < w(i_m)$.
また，$C' = \sum_{t=1\sim s} w(j_t) + C''$　だから,
　$val(Opt) < \sum_{j=1\sim k} v(i_j) + \sum_{j=k+1\sim m-1} v(i_j) + (\sum_{t=1\sim s} w(j_t) + w(i_m)) \times v(i_m)/w(i_m)$
　　$\leqq \sum_{j=1\sim k} v(i_j) + \sum_{j=k+1\sim m-1} v(i_j) + \sum_{t=1\sim s} v(j_t) + v(i_m)$
　　$\leqq val(S) + v(i_m) \leqq val(S^*) + val(Opt)/(k+1)$.

By the property of Greedy Algorithm, the objects added in S have larger values per unit weight than the object i_m (otherwise i_m should have been added) and so
　$v(j_t)/w(j_t) \geqq v(i_m)/w(i_m)$, $t=1, \ldots, s$.

Letting S* be an approximate solution obtained by the algorithm, we have
　$val(S^*) \geqq val(S) = \sum_{j=1\sim k} v(i_j) + \sum_{j=k+1\sim m-1} v(i_j) + \sum_{t=1\sim s} v(j_t)$,
where the remaining capacity C' for the part up i_m to in Opt and the remaining capacity C" in S are given by
　$C' = C - \sum_{j=1\sim k} v(i_j) - \sum_{j=k+1\sim m-1} v(i_j)$,
　$C'' = C' - \sum_{t=1\sim s} v(j_t)$



Since we have larger value if we fill in the remaining capacity C' by the value $v(i_m)/w(i_m)$, we have
　$val(Opt) \leqq \sum_{j=1\sim k} v(i_j) + \sum_{j=k+1\sim m-1} v(i_j) + C' v(i_m)/w(i_m)$.
Since i_m is not in S and there is no capacity to be included in S,
　$C'' < w(i_m)$.
Also, since $C' = \sum_{t=1\sim s} w(j_t) + C''$　we have
　$val(Opt) < \sum_{j=1\sim k} v(i_j) + \sum_{j=k+1\sim m-1} v(i_j) + (\sum_{t=1\sim s} w(j_t) + w(i_m)) \times v(i_m)/w(i_m)$
　　$\leqq \sum_{j=1\sim k} v(i_j) + \sum_{j=k+1\sim m-1} v(i_j) + \sum_{t=1\sim s} v(j_t) + v(i_m)$
　　$\leqq val(S) + v(i_m) \leqq val(S^*) + val(Opt)/(k+1)$.

結局，次式を得る：

$val(Opt) \leqq val(S^*)+val(Opt)/(k+1)$.

これより，

$$\frac{val(Opt)}{val(S^*)} \leqq 1+1/k \leqq 1+\varepsilon$$

すなわち，アルゴリズムによって得られる近似解S*の近似率は高々εである．

計算時間

計算時間は$O(n^{k+1})$であるが，**$\varepsilon \geqq 1/k$**を満たす最小の正整数をkとしたから，1/εに関しては**指数関数**となっている．したがって，

多項式時間近似方式(PTAS)ではあるが，
完全多項式時間近似方式(FPTAS)ではない．

では，ナップサック問題に対して完全多項式時間近似方式は存在するか？

Finally, we have：

$val(Opt) \leqq val(S^*)+val(Opt)/(k+1)$.

This leads to

$$\frac{val(Opt)}{val(S^*)} \leqq 1+1/k \leqq 1+\varepsilon$$

That is, the approximation ratio of a solution S* obtained by the algorithm is at most ε.

Computation time

Computing time is $O(n^{k+1})$. Since k is the least positive integer satisfying **$\varepsilon \geqq 1/k$**, it is **exponential in** 1/ε. Thus,

It is a Polynomial-Time Approximation Scheme(PTAS), but it is not a Fully Polynomial-Time Approximation Scheme(FPTAS).

Then, is there any fully polynomial-time approximation scheme for the knapsack problem?

ナップサック問題に対する完全多項式時間近似方式

目標：計算時間を入力サイズnと1/εに関して多項式にすること
考え方：荷物の重さが整数で与えられる場合に最適解を求める動的計画法のアルゴリズムを利用．

アルゴリズムP33-A3: 完全多項式時間近似方式

(1) 得たい相対誤差εに対して，$K= \varepsilon v_{max}/n$とおく．
ただし，v_{max}は最大の価値，$v_{max} = \max\{v_i, i=1, \ldots, n\}$．
(2) それぞれの荷物iの価値を$v'_i =[v_i /K]$とする．[]は切り捨て．
(3) 上で変換した価値に対して，動的計画法を用いて最適解Sを求め，近似解として出力する．

補題14-1：アルゴリズムP33-A3で求められる解の相対誤差はεである．

補題14-2：アルゴリズムP33-A3の計算時間はnと1/εのいずれに関しても多項式である．

FPTAS for Knapsack Problem

Goal：polynomial time algorithm both in input size n and 1/ε
Idea: To use an algorithm based on dynamic programming that finds an optimal solution for integer weight case.

Algorithm P33-A3: FPTAS

(1) For a relative error εto achieve, let $K= \varepsilon v_{max}/n$.
Here, v_{max} is the largest value, $v_{max} = \max\{v_i, i=1, \ldots, n\}$.
(2) Change the value of object i to $v'_i =[v_i /K]$. []: floor function.
(3) By dynamic programming, find an optimal solution S for the modified values and output it as an approximate solution.

Lemma 14-1：The relative error of the solution obtained in Algorithm P33-A3 is ε.

Lemma 14-2：The computation time of Algorithm P33-A3 is polynomial both in n and 1/ε.

補題14-1：アルゴリズムP33-A3で求められる解の相対誤差はεである．

証明：アルゴリズムで得られる解をSとしよう．
Sは近似した問題の最適解なので，

$\sum_{i \in S} v'_i \geqq \sum_{i \in Opt} v'_i$

が成り立つ．一方，各iについて，$v'_i =[v_i /K]$だから，

$v_i \geqq K v'_i \geqq v_i - K$

が成り立つ．よって，次式を得る．($K= \varepsilon v_{max}/n$に注意）

$val(S) = \sum_{i \in S} v_i \geqq \sum_{i \in S} Kv'_i \geqq \sum_{i \in Opt} v_i - nK$
$= val(Opt) - \varepsilon v_{max} \geqq (1-\varepsilon)val(Opt)$.

したがって，

$[val(Opt) - val(S)] / val(Opt)$
$\leqq [val(Opt)-(1-\varepsilon)val(Opt)]/val(Opt)=[1-(1-\varepsilon)]/1=\varepsilon$

を得る．

証明終

Lemma 14-1：The relative error of the solution obtained in Algorithm P33-A3 is ε.

Proof：Let S be a solution obtained by the algorithm.
Since S is an optimal solution to the approximated problem, we have

$\sum_{i \in S} v'_i \geqq \sum_{i \in Opt} v'_i$

On the other hand, $v'_i =[v_i /K]$ for each I and we have

$v_i \geqq K v'_i \geqq v_i - K$.

Thus, we have (note $K= \varepsilon v_{max}/n$)

$val(S) = \sum_{i \in S} v_i \geqq \sum_{i \in S} Kv'_i \geqq \sum_{i \in Opt} v_i - nK$
$= val(Opt) - \varepsilon v_{max} \geqq (1-\varepsilon)val(Opt)$.

Hence, we have,

$[val(Opt) - val(S)] / val(Opt)$
$\leqq [val(Opt)-(1-\varepsilon)val(Opt)]/val(Opt)=[1-(1-\varepsilon)]/1=\varepsilon$

Q.E.D.

補題14-2:アルゴリズムP33-A3の計算時間はnと1/εのいずれに関しても多項式である.

この補題を証明するために,動的計画法のアルゴリズムを示そう.

仮定:荷物の価値はすべて正整数とし,その和をVとする.
n行V列の表W[]を用意する.

W[i,v] = {1, ... , i}の部分集合で,その価値の和がちょうどvになるものの中で,その重さの和が最小になるものをS(i,v)とするとき,S(i,v)の重さ.
ただし,重さの和がちょうどvになる荷物の組合せがなければ,W[i,v]=∞.

このとき,

$W[i,0]=0, \quad i=1, \ldots, n$
$W[1,v_1] = w_1,$
$W[1,v] = \infty, \; v \neq v_1$

漸化式は次の通り

$W[i+1,v]=\min\{ W[i,v], w_{i+1} +W[i, v- v_{i+1}]\}$ 　 $v_{i+1} \leqq v$のとき
$= W[i,v]$ 　 $v_{i+1} > v$のとき

これで最適解が求まる.計算時間は明らかに O(nV).
アルゴリズムP33-A3の計算時間は,

$O(n \sum v'_i)=O(n^2 v'_{max})=O(n^2[n/\varepsilon])$

となり,補題を得る.

Lemma 14-2: The computation time of Algorithm P33-A3 is polynomial both in n and 1/ε.

We shall show an algorithm using dynamic programming to prove the lemma.

Assumption: Each object has positive value and the total sum is V.
Prepare a tablel W[] with n rows and V columns.

W[i,v] = the weight of S(i,v) where S(i, v) is a subset of {1, ... , i} and the sum of values of objects in it is exactly v. W[i,v]=∞ if there is no such subset.

Then,

$W[i,0]=0, \quad i=1, \ldots, n$
$W[1,v_1] = w_1,$
$W[1,v] = \infty \; v \neq v_1$

The recurrence equation is as follows:

$W[i+1,v]=\min\{ W[i,v], w_{i+1} +W[i, v- v_{i+1}]\}$ 　 if $v_{i+1} \leqq v$
$= W[i,v]$ 　 if $v_{i+1} > v$

In this way we can find an optimal solution. The computation time is obviously O(nV). Thus, the algorithm P33-A3 takes time

$O(n \sum v'_i)=O(n^2 v'_{max})=O(n^2[n/\varepsilon])$

which proves the lemma.

演習問題E14-2: 動的計画法のアルゴリズムを正確に記述し,それに基づいてプログラムを作成せよ.

演習問題E14-3: 辺に重みのついたグラフ上で,重み最小の巡回路(各頂点を少なくとも1度通る最短経路)を求める問題はNP完全であるが,この問題に対する2-近似アルゴリズムを示せ.

演習問題E14-4: 平面上に置かれたn個の点をすべて通る最短巡回路を求めるユークリッド行商人問題もNP完全であるが,この問題に対する1.5-近似のアルゴリズムを示せ.

Exercise E14-2: Describe the algorithm using dynamic programming and write a program based on it.

Exercise E14-3: The problem if finding a least-weight tour (a shortest path visiting every vertex at least once) in a weighted graph is NP-complete. Give a 2-approximation algorithm for the problem.

Exercise E14-4: The Euclidean Traveling Salesperson problem of finding a shortest tour through all the points place in the plane is also NP-complete. Give a 1.5-approximation algorithm for this problem.