ALMA
ALGORITHMS

Fall 2016

Ioannis Milis

Approximation Algorithms PTASs and FPTASs

Approximations: Good, better, best and more …

Non - constant approximation : $C/OPT \leq f(n)$

Constant (ρ-)approximation : $C/OPT \leq \rho$ (a constant, e.g. 3/2)

Polynomial Time Approximation Schemes (PTAS)

- $C/OPT \leq 1 + \varepsilon$, for any $\varepsilon > 0$
- O(poly (|I|)), O(exp (1/ε)), e.g. $O(n^{3/\varepsilon})$

Fully Polynomial Time Approximation Schemes (FPTAS)

- $C/OPT \leq 1 + \varepsilon$, for any $\varepsilon > 0$
- O(poly (|I|)), O(poly (1/ ε)) !!! e.g. $O((1/\varepsilon)^2 n^3)$

Additive approximation

- $C \leq OPT + f(n)$ or $C \leq OPT + k$ (a constant), e.g. $C \leq OPT+1$!

Partitions of weighted sets

SUBSET SUM

I: objects S={1,…,n}, positive integer weights w_i , i=1,…,n, positive integer W

Q: is there $A \subseteq S$ s.t. $\sum_{i \in A} w_i = W$?

PARTITION

I: objects S={1,…,n}, positive integer weights w_i , i=1,…,n

Q: is there $A \subseteq S$ s.t $\sum_{i \in A} w_i = \sum_{i \in S-A} w_i \left(= \frac{1}{2} \sum_{i \in S} w_i\right)$?

0-1 KNAPSACK

I: objects S={1,…,n}, positive integer weights w_i , i=1,…,n,

values v_i , i=1,…,n, positive integer W

Q: find $A \subseteq S$ s.t. $\sum_{i \in A} w_i \leq W$ and $\sum_{i \in A} v_i$ is maximized.

Partitions of weighted sets

BIN PACKING

I: objects S={1,…,n}, positive integer weights w_i , i=1,…,n, positive integer W

Q: find a partition of S into $A_1, \dots, A_m$ s.t. $\sum_{i \in A_j} w_i \leq W, j = 1,2,\dots,m$
and m is minimized

SCHEDULING (P||C_{max})

I: objects S={1,…,n}, positive integer weights w_i , i=1,…,n, positive integer m

Q: find a partition of S into $A_1, \dots, A_m$ s.t. $\max_{1 \leq j \leq M} \{ \sum_{i \in A_j} w_i \}$ is minimized

Knapsack problems

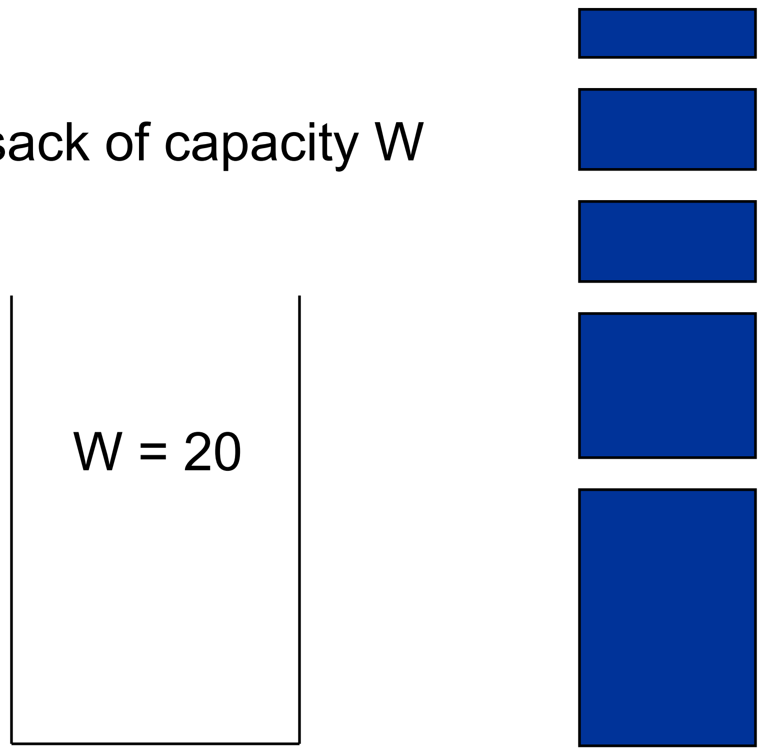
- We are given a knapsack with maximum capacity W, and a set $S=\{1,2,\ldots,n\}$ of n items

- Each item i has weight w_i and value v_i (all w_i, v_i and W are integers)

Problem: How to pack the knapsack to achieve maximum total value of packed items?

Knapsack problems

Knapsack of capacity W



Items	Weight w_i	Value v_i
	2	3
	3	4
	4	5
	5	8
	9	10

Knapsack problems

Three (basic) versions of the problem:

1. Fractional knapsack

   Items are divisible: take any fraction of an item

   O(poly) by a greedy algorithm

2. 0-1 knapsack

   Items are indivisible: take an item or not

   NP-complete, O(nW), by a DP algorithm

3. Integer knapsack

   Multiple copies of indivisible items:

   take any number of copies of an item

   NP-complete, O(nW), by a DP algorithm

Knapsack problems

Fractional knapsack

$$\max \sum_{i \in S} v_i\, x_i, \text{ s.t. } \sum_{i \in S} w_i x_i \le W, \text{ and } x_i \in [0,1]$$

0-1 Knapsack

$$\max \sum_{i \in S} v_i\, x_i, \text{ s.t. } \sum_{i \in S} w_i x_i \le W, \text{ and } x_i \in \{0,1\}$$

Integer Knapsack

$$\max \sum_{i \in S} v_i\, x_i, \text{ s.t. } \sum_{i \in S} w_i x_i \le W, \text{ and } x_i \in N$$

Fractional Knapsack

Greedy algorithm:

take the item with the maximum value per unit (v_i/w_i) among the remaining items, as much as the capacity of the knapsack allows

Note: knapsack is loaded by the whole items, but the last

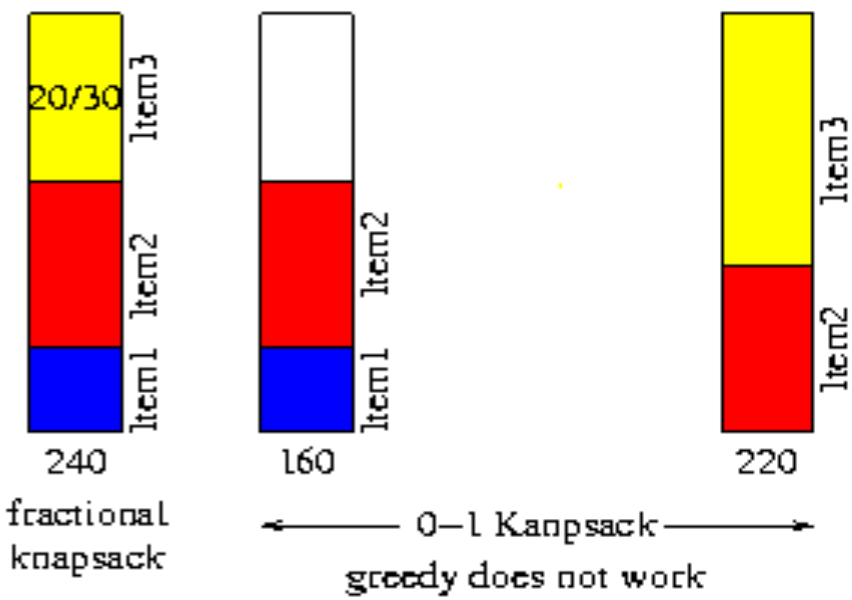
Q: Prove that the greedy algorithm is optimal

Complexity: O(n logn) – why?

Fractional vs 0-1 Knapsack

Let capacity knapsack be $W = 50kg$. Let there be 3 items.

Item	Weight	Value
1	10	60
2	20	100
3	30	120



0-1 Knapsack vs SUBSET SUM

0-1 KNAPSACK (DECISION)

I: objects S={1,…,n}, positive integer weights w_i , i=1,…,n,
values v_i , i=1,…,n, positive integer W, positive integer V

Q: Is there $A \subseteq S$ s.t. $\sum_{i \in A} w_i \le W$ and $\sum_{i \in A} v_i \ge V$.

Let an instance of 0-1 KNAPSACK where : $w_i = v_i$, $1 \le i \le n$, and $W = V$

Then, the Question becomes:

Q: Is there $A \subseteq S$ s.t. $\sum_{i \in A} w_i \le W$ and $\sum_{i \in A} v_i \ge V$, that is

Q: Is there $A \subseteq S$ s.t. $\sum_{i \in A} w_i \le W$ and $\sum_{i \in A} w_i \ge W$, that is

Q: Is there $A \subseteq S$ s.t. $\sum_{i \in A} w_i = W$, that is SUMSET-SUM

Hence, 0-1 KNAPSACK is a generalization of SUBSET SUM

0-1 Knapsack

Brute-force approach

- there are 2^n possible combinations of n items
- Go through all combinations and find the one with the most total value and with total weight less or equal to W
- Running time: $O(n2^n)$

Can we do better?

- Yes, with a DP algorithm

DP for 0-1 Knapsack

Subproblem:

V[k,w] : the subproblem with S_k={1, 2, …, k} items and capacity w

Item k either can be in the optimal solution to V[k,w] or not.

- First case: $w_k > w$.
  - item k can not be in the optimal solution
  - V[k,w] = V[k-1, w]

- Second case: $w_k \leq w$.
  - item k can be in the optimal solution
  - the best solution for V[k,w] is one of the next two:
    - the best solution for V[k-1, w] or
    - the best solution for V[k-1, $w-w_k$] plus the value of item k:

      V[k-1, $w-w_k$] $+v_k$
  - V[k,w] = max { V[k-1, w], V[k-1, $w-w_k$] $+v_k$ }

DP for 0-1 Knapsack

Recursive Formula

$$V[k,w] = \begin{cases} 0 & \text{if } k = 0 \text{ or } w = 0 \\ V[k-1,w] & \text{if } k, w \geq 1 \text{ and } w_k > w \\ \max\{V[k-1,w], V[k-1,w-w_k]+v_k\} & \text{if } k, w \geq 1 \text{ and } w_k \leq w \end{cases}$$

DP for 0-1 Knapsack

```
0-1 Knapsack Value (w,v,W)
{
for w := 0 to W do  V[0,w] := 0;
for k = 0 to n do
{  V[k,0] := 0
   for w := 1 to W do
     if wk ≤ w            // item i can be in the solution
       then  V[k,w]:= max { vk + V[k-1,w-wk], V[k-1,w]}
       else  V[k,w]:= V[k-1,w]     }       // wk > w
}
```

Complexity O(nW)

Another DP for 0-1 Knapsack

Previous Algorithm:

OPT = V[n,W] can be found in O(nW) time

Another Algorithm:

OPT can be found in O(n OPT), that is $O(n^2 v_{max})$ time (why?)

Subproblem:

C[k,v] : the minimum capacity yielding a value v using items 1,2,…,k

OPT = maximum v for which C[n,v] ≤ W

Another DP algorithm for 0-1 knapsack

Subproblem:

C[k,v]: the smallest capacity yielding a value v using items 1,2,…,k

C[k,0] = 0, k=1,2,…,n

The optimal solution to C[k,v] either contains item k or not.

- First case: $v_k > v$.
  - item k can not be part of the solution
    - it yields to a total value > v, which is unacceptable
  - C[k,v] = C[k-1, v]

- Second case: $v_k \leq v$.
  - item k can be in the solution
  - the best solution for C[k,v] is one of the two:
    - the best solution for C[k-1, v] or
    - the best solution for $C[k-1, v-v_k]$ plus the weight of item k
  - $C[k,v] = \min\{C[k-1,v], C[k-1, v-v_k] + w_k\}$

Another DP algorithm for 0-1 knapsack

What values C[k,v] we have to calculate?

Let OPT be the (value of the) optimal solution

$v_{max} = \max_k \{ v_k \}$ OPT $\leq n\, v_{max}$

Calculate the values C[k,v], $0 \leq k \leq n$, $0 \leq v \leq n\, v_{max}$

k \ v	0	1	2	3	…	…	$n\ v_{max}$
1	0	↓	↓	↓			↓
2	0						
3	0						
.	0						
.	0						
n	0						

Another DP algorithm for 0-1 knapsack

Which C[k,v] corresponds to the optimal solution ?

OPT = maximum v for which C[n, v] $\leq$ W

Complexity $O(n^2 v_{max})$

k \ v	0	1	2	3	…	…	$n\, v_{max}$
1	0						
2	0						
3	0						
.	0						
.	0						
n	0						

A FPTAS for 0-1 KNAPSACK

Max $\Sigma_i \, v_i x_i$ = OPT (optimal value)

such that : $\Sigma_i \; x_i w_i \leq W$

and $x_i \in \{0, 1\}, \; 0 \leq i \leq n$

There is an $O(n^2 \, v_{max})$ DP algorithm for 0-1 KNAPSACK

Recall that $v_{max} = \max_i \{v_i\}$ and $v_{max} \leq OPT \leq n v_{max}$

A FPTAS for 0-1 KNAPSACK

SCALED PROBLEM

Scale all items values by k, i.e. $v_j(k) = \lfloor v_j / k \rfloor$

Algo-S

{ Solve the scaled problem by the last DP algorithm;

Let $S(k) \subseteq \{1,2,\ldots,n\}$ be the optimal solution to the scaled problem;

Return the value of this solution for the original problem; }

A FPTAS for 0-1 KNAPSACK

Algo-S is a FPTAS for 0-1 KNAPSACK

Proof:

Optimal solution of the original problem: S* $\subseteq$ {1,2,…,n} of value OPT

Optimal solution of the scaled problem: S(k) $\subseteq$ {1,2,…,n} of value OPT(k) for the original problem

* S(k) is of greater value than any solution (and S*) for the scaled problem

$$\frac{v_j}{k} - 1 \overset{(1)}{<} v_j(k) = \left\lfloor \frac{v_j}{k} \right\rfloor \overset{(2)}{\le} \frac{v_j}{k}$$

$$OPT(k) = \sum_{j \in S(k)} v_j \overset{\text{by (2)}}{\ge} \sum_{j \in S(k)} k\, v_j(k)$$

$$= k \sum_{j \in S(k)} v_j(k) \overset{\text{by *}}{\ge} k \sum_{j \in S^*} v_j(k) =$$

$$\overset{\text{by (1)}}{\ge} k \sum_{j \in S^*} \left(\frac{v_j}{k} - 1\right) = \sum_{j \in S^*} v_j - k \sum_{j \in S^*} 1 = OPT - k|S^*|$$

$$\ge OPT - kn, \text{ since } |S^*| \le n$$

$$\Rightarrow OPT(k) \ge OPT - kn$$

A FPTAS for 0-1 KNAPSACK

Proof (cont.):

$$OPT(k) \geq OPT - n \cdot k,$$

$$\text{Choose} \quad k = \frac{\varepsilon \cdot v_{\max}}{n} \leq \frac{\varepsilon \cdot OPT}{n}, \text{since } v_{\max} \leq OPT$$

$$\text{Hence,} \; OPT(k) \geq OPT - \varepsilon \cdot OPT = (1-\varepsilon)OPT$$

Complexity

$$O\left(n^2 \left\lfloor \frac{v_{\max}}{k} \right\rfloor\right), \text{that is} \quad O(n^3 \frac{1}{\varepsilon}), \quad \text{since} \left\lfloor \frac{v_{\max}}{k} \right\rfloor = \frac{n}{\varepsilon}$$

O(poly(n))

O(poly(1/ε)) A FPTAS !

Strong NP-completeness and pseudo-polynomial algorithms

A problem Π is strongly NP-complete if

- it remains NP-complete even if any instance of length |I| is restricted to contain integers at most O(poly(|I|)) or
- it remains NP-complete even if its instances are coded in unary

0-1 Knapsack is NOT strongly NP-complete but it is NP-complete.
There is an O(nW) algorithm; it is O(poly) if W is O(poly(|I|))

Let N(I) be the largest number appearing in an instance of a problem

- An algorithm is a pseudo-polynomial one if it is polynomial in |I| and N(I)
- Unless P=NP, there is no pseudo-polynomial algorithm for strongly NP-complete problems (next slide)
- For problems that are NP-complete, but not strongly NP-complete there is a pseudo-polynomial algorithm (usually a dynamic programming one)

Strong NP-completeness and pseudo-polynomial algorithms

Let : I be an instance of a problem Π, of size $|I|$
N(I) be the largest number in I
p(n) be a polynomial
$\Pi_{p(n)}$ be Π restricted to instances for which $N(I) \leq p(|I|)$

We say that Π is strongly NP-complete if $\Pi_{p(n)}$ is NP-complete

Th. Unless P=NP, there is no pseudo-polynomial algorithm for a strongly NP-complete problem Π

Proof:

Suppose that there exists such a pseudo-polynomial algorithm Q for Π

Q solves any instance of Π in $q(|I|, N(I))$ time; q: a polynomial

Q solves $\Pi_{p(n)}$ in $q(|I|, p(|I|))$ time, that is polynomial in $|I|$

P=NP !

Strong NP-completeness and FPTASs

Let :

Π be a strongly NP-complete optimization problem (its decision version)

I be an instance of Π, of size |I|

N(I) be the largest number in I

p(n) be a polynomial

All the values in the input and output of Π are integers

For any instance of Π it holds that $C^* \leq p(N(I))$

Th. Unless P=NP, there is no an FPTAS for Π

Proof:

- Suppose that there an FPTAS, F, for Π
- Apply this to Π with $\varepsilon = 1 / (p(N(I)) +1)$
- $(|C^*-C|)/C^* \leq \varepsilon \Rightarrow |C^*-C| \leq \varepsilon C^* = C^* / (p(N(I)) +1) < 1$, since $C^* \leq p(N(I))$
- F solves Π exactly ! (since all feasible solutions are integers)
- F takes q(|I|, 1/e) time; q: polynomial,

  that is q(|I|, p(N(I)) +1)) time, a polynomial in both |I| and N(I) !
- F is pseudo-polynomial algorithm for Π !
- P=NP !

BIN PACKING

Approximations for BIN-PACKING

Alternative instance for BIN-PACKING:

M=1, $w_i \in (0,1]$ **for** $1 \le i \le n$.

Q: Can S be packed into B bins ?

BIN-PACKING is (weakly) NP-complete since PARTITION is a special case of BIN-PACKING for $B = 2$ **and** $M = \frac{1}{2}\sum_{i \in S} w_i$

However, we know that BIN PACKING is stronly NP-complete

Approximation algorithms for BIN-PACKING:

- **NEXT-FIT:** $m \le 2OPT$ **(tight)**
- **FIRST-FIT:** $m \le 1.7OPT$ **(tight)**
- **and many other …**

Approximations for BIN-PACKING

Unless $P \neq NP$, there is no $\left(\frac{3}{2} - \delta\right)$ -approximation algorithm for BIN-PACKING

<u>Proof:</u>

Assume that there is an algorithm A such that $m \leq \left(\frac{3}{2} - \delta\right) OPT$.

Run A for $M = \frac{1}{2} \sum_{i \in S} w_i$

If m=2 then PARTITION has answer YES!

If m $\geq$ 3 we have

$$m < \frac{3}{2} OPT \Rightarrow OPT > \frac{2}{3} m \geq \frac{2}{3} \cdot 3 = 2$$

So OPT>2 and thus, PARTITION has answer NO!

Hence, we have an O(poly) algorithm for PARTITION,

That is P=NP, a contradiction.

What if OPT increases with n?

APTAS for BIN-PACKING

An Asymptotic PTAS (APTAS) produces a (1+ε)-approximate solution, that is, for each ε>0, there is N>0 such that the APTAS has an approximation ratio 1+ε for all instances having OPT≧N.

There is an APTAS for BIN-PACKING.

Three steps:

(1) Instances with fixed number, k, of items sizes

optimal in O(poly|I|)

(2) Instances with items of size s $w_i \geq \varepsilon$

(1+ε)-approximation in O(poly|I|)

(3) First-Fit for items of sizes $w_i < \varepsilon$

(1+2ε)OPT + 1 approximation in O(poly|I|)

BIN-PACKING: fixed # of item sizes

Instance: $(n_1, \dots, n_k)$, n_j : # of items of size j, $0 \le j \le k$. ($\sum_{j=1}^{k} n_j = n$)

Consider a k-tuple $(i_1, \dots, i_k)$, $0 \le i_j \le n_j$, $1 \le i \le k$.

$(i_1, \dots, i_k) \subset \{0,1,\dots,n_1\} \times \{0,1,\dots,n_2\} \times \dots \times \{0,1,\dots,n_k\} = A$

$|A| = O(n^k)$

$BINS(i_1, \dots, i_k)$: min # of bins required to pack those $\sum_{j=1}^{k} i_j$ items.

$Q \subset A$: all k-tuples such that $BINS(q_1, \dots, q_k) = 1$, $0 \le q_j \le n_j$, $1 \le j \le k$

i.e. the $\sum_{j=1}^{k} q_j$ items can be packed in one bin.

$Q \subset A \Rightarrow |Q| \sim O(n^k)$

BIN-PACKING: fixed # of item sizes

- Find Q in $O(n^k)$ time, O(poly) as k is fixed.

- Fill the k-dimensional table $BINS(i_1, \dots, i_k)$

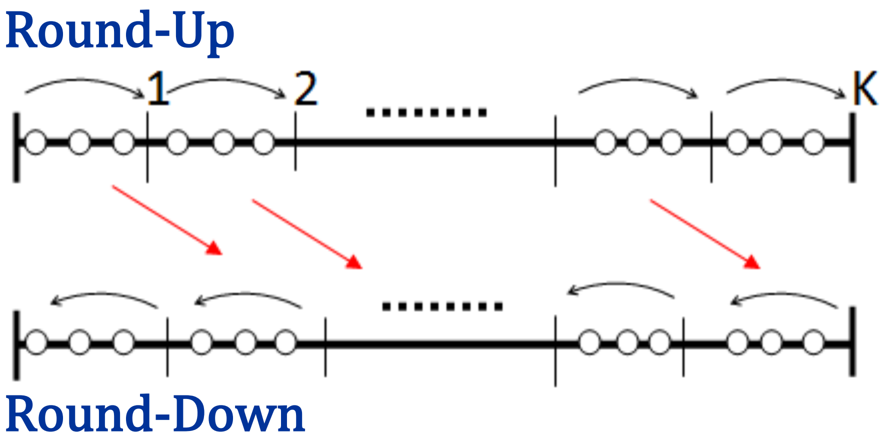
- Recurrence (DP):

$$BINS(i_1, \dots, i_k) = 1 + \min_Q\{BINS(i_1 - q_1, \dots, i_k - q_k)\}$$

$$BINS(q_1, \dots, q_k) = 1, \forall (q_1, \dots, q_k) \in Q$$

- Return $BINS(n_1, \dots, n_k)$ in $O(n^{2k})$ time.

BIN-PACKING: sizes at least ε



Sort items in non-decreasing order.
Partition items into $K = \left\lceil \frac{1}{\varepsilon^2} \right\rceil$ groups each of them having at most $Q = \lfloor n\varepsilon^2 \rfloor$ items.
(Two groups may contain items of the same size.)

Pack I^{up} (fixed number $K = \left\lceil \frac{1}{\varepsilon^2} \right\rceil$ of object sizes)
In $O(n^{2k})$ time, that is $O(n^{2/e^2})$.

The algorithm returns a solution OPT(I^{up}).
Return this packing for instance I (this is a feasible packing).

BIN-PACKING: sizes at least ε

Analysis:

1) $OPT(I) \geq n\,\varepsilon$, standard lower bound

2) $OPT\,(I^{down}) \leq OPT(I)$, all items have smaller sizes

3) $OPT(\,I^{down}\,)$ is also a packing for I^{up}, **but for Q largest items.**

Hence, $OPT(I^{up}) \leq OPT(I^{down}) + Q \leq OPT(I) + Q$

Since, $OPT(I) \geq n\,\varepsilon$ and $Q = \lfloor n\varepsilon^2 \rfloor$ we get $Q \leq \varepsilon\, OPT$

Therefore: $OPT(I^{up}) \leq (1+\varepsilon)OPT(I)$

APTAS for BIN-PACKING

I: original instance of the problem.

1. Ignore items of sizes $w_i < \varepsilon$ (instance I')
2. Construct instance $I^{up'}$
3. Find an optimal packing for this instance $OPT(I^{up'})$
4. Return this packing for original items in I'
5. Pack items of sizes $w_i < \varepsilon$ using First-Fit on this packing.

APTAS for BIN-PACKING

Let M the total number of bins used after First-Fit.

- If no additional bins are needed.

$$M = OPT\left(I^{up'}\right) \le (1+\varepsilon)OPT(I') \le (1+\varepsilon)OPT(I)$$

- If additional bins are needed.

All, but the last, bins are full to at least 1-ε.

Thus,

$$\left.\begin{array}{c}\sum w_i \ge (M-1)(1-\varepsilon) \\ OPT \ge \sum w_i\end{array}\right\} \Rightarrow OPT \ge (M-1)(1-\varepsilon) \Rightarrow$$

$$\Rightarrow M \le \left(\frac{1}{1-\varepsilon}\right)OPT + 1 \Rightarrow$$

$$\Rightarrow M \le (1+2\varepsilon)OPT + 1, \ for \ \ \varepsilon \le 1/2$$