A Graduate Course on Algorithms **Spring Semester, 1997/8**

Lecture 2: Classical Results (cont.)

Lecturer: Uri Zwick *Scribe: Manor Mendel*

In this lecture we continue our survey of classical combinatorial approximation results. In the process we also present some easy applications of general techniques used in the design of combinatorial approximation algorithms.

2.1 LP Relaxation for the Hitting Set Problem (HS)

> In the *hitting set* problem we are given n subsets $S_1, \ldots, S_n$ of $X = \{1, \ldots, m\}$ and we are required to find a subset $C \subseteq X$ whose size is as small as possible such that $C \cap S_i \neq \phi$, for all $1 \leq i \leq n$.

This is a simple restatement of the set cover problem presented in the previous lecture, with the elements and the sets switching parts. The greedy algorithm achieves therefore a performance ratio of $H(\Delta)$, where Δ this time is the maximum degree of the elements of X, where the degree of an element $x \in X$ is the number of subsets among $S_1, \ldots, S_n$ to which it belongs.

We present here another approximation algorithm for the hitting set problem. The performance guarantee of this algorithm is slightly inferior to the performance guarantee of the greedy algorithm. It presents, however, a simple example of the use of linear programming relaxations and of randomized rounding.

An instance of HS can be easily cast as an integer programming problem (IP):

$$\min \sum_{i=1}^{m} x_i$$

$$\sum_{i \in S_j} x_i \geq 1 \quad , \; 1 \leq j \leq n \tag{2.1}$$

$$x_i \in \{0,1\} \quad , \; 1 \leq i \leq m \tag{2.2}$$

If $x_1, \ldots, x_n$ is an optimal solution of this program then $C = \{i \mid 1 \leq i \leq m \text{ and } x_i = 1\}$ is an optimal cover (or hitting set, if you prefer).

The linear programming *relaxation* of the above integer program is the linear program (LP) obtained by replacing constraints (2.2) with

$$0 \leq x_i \leq 1 \quad , \; 1 \leq i \leq m \tag{2.3}$$

(Note that the upper bounds in constraints (2.3) is redundant.)

Let $\hat{x}_1, \ldots, \hat{x}_m$ be an optimal solution of the linear program. Such an optimal solution can be found in polynomial using several available polynomial time algorithms. We will make a heavy use of this fact in this course.

Any feasible solution of the integer program (IP) is also a feasible solution of the linear program. Thus $\sum_{i=1}^{m} \hat{x}_i$ is a *lower bound* on the size of the optimal hitting set C^*.

We now define the following integer "solution" $x_1, \ldots, x_m$: for every $1 \leq i \leq m$, x_i is independently assigned the value 1 with probability $\hat{x}_i$, and the value 0 with the complementary probability $1-\hat{x}_i$. We then let $C = \{i \mid x_i = 1\}$. The (random) set C obatained is not necessarily a legal solution. Some of the sets $S_1, S_2, \ldots, S_n$ may not be hit by C.

We want to obtain an upper bound on the probability that a set S_j is *not* hit by C. Assume $|S_j| = k$. From constrains (2.1), we get that $\sum_{i \in S_j} \hat{x}_i \geq 1$. It is easy to check that this implies that $\prod_{i \in S_j}(1-\hat{x}_i) \leq (1-\frac{1}{k})^k$. Thus,

$$\Pr[C \cap S_j = \phi] = \prod_{i \in S_j}(1-\hat{x}_i) \leq \left(1-\tfrac{1}{k}\right)^k < \mathrm{e}^{-1} .$$

If we take the union of $\ln(2n)$ randomly chosen sets generated in a similar manner to C, then the probability that certain set S_j is not hit is at most $(e^{-1})^{\ln 2n} = 1/2n$. The probability that *any* set, among $S_1, S_2, \ldots, S_n$, is not hit is at most 1/2.

Choosing $t = \ln(2n)$ such random sets is equivalent to assigning x_i the value 1 with probability $1-(1-\hat{x}_i)^t$, and then taking a single set $C = \{i \mid 1 \leq i \leq m\}$.

The expected size of each C is $\sum_{i=1}^m \hat{x}_i$. We saw earlier that $\sum_{i=1}^m \hat{x}_i \leq |C^*|$, where C^* is an optimal solution. Thus, our approximation algorithm achieves an *expected* performance ratio of $O(\log n)$.

We can do slightly better than this. Assume now that each element is contained in at most Δ sets. It is easy to see that if $(\hat{x}_1, \ldots, \hat{x}_m)$ is an optimal solution of the linear program then $|C^*| \geq \sum_{i=1}^m \hat{x}_i \geq \frac{n}{\Delta}$. We can obtain an expected approximation ratio of $\ln\Delta + 1$ using the following approach. Make only $\ln\Delta$ random choices. Each set S_j is not hit with a probability of at least $1/\Delta$. The expected number of elements that were not hit is therefore at most n/Δ. Now, simply choose an arbitrary element of each set that was not hit. The expected number of elements in the hitting set constructed is at most

$$\ln\Delta \cdot |C^*| + \frac{n}{\Delta} \leq (\ln\Delta + 1) \cdot |C^*| .$$

This is almost as good as the greedy algorithm described in the previous lecture. Note, however, the new algorithm needs to solve a linear program, which is not a trivial task, and that it is randomized.

As mentioned in the previous lecture, Feige [] has recently shown that no polynomial time approximation algorithm may achieve a ratio of less than $\ln m$ for the hitting set problem, unless P=NP. (Note that m here is the number of *sets*, which is equivalent to the number of *elements* in the set cover problem.)

2.2 The Vertex Cover Problem

The Vertex Cover problem is defined as follows:

> Given an undirected graph $G = (V, E)$, find a set $C \subseteq V$ of minimal size such that $C \cap e \neq \phi$, for every $e \in E$. Such a set C is said to be a minimum vertex cover, i.e., a set of vertices that covers all the edges.

The vertex cover is a special case of the hitting set problem in which each set that we are supposed to hit, i.e., each edge, is of size exactly two. We can use the greedy algorithm for set cover to get a performance ratio of $H(\Delta) = \sum_{i=1}^{\Delta} \frac{1}{\Delta}$, where Δ is the maximum degree in G. The performace ratio of the greedy algorithm may be as bad as $\ln n$ is the graph $G = (V, E)$ contains high degree vertices. Figure 2.1 shows that for every $\Delta \geq 1$ there are graphs with maximal degree Δ in which the vertex cover returned by the greedy algorithm may be $H(\Delta)$ times larger than the minimum vertex cover. We will shortly see that a much better performance ratio ratio may be obtained for the vertex cover problem.

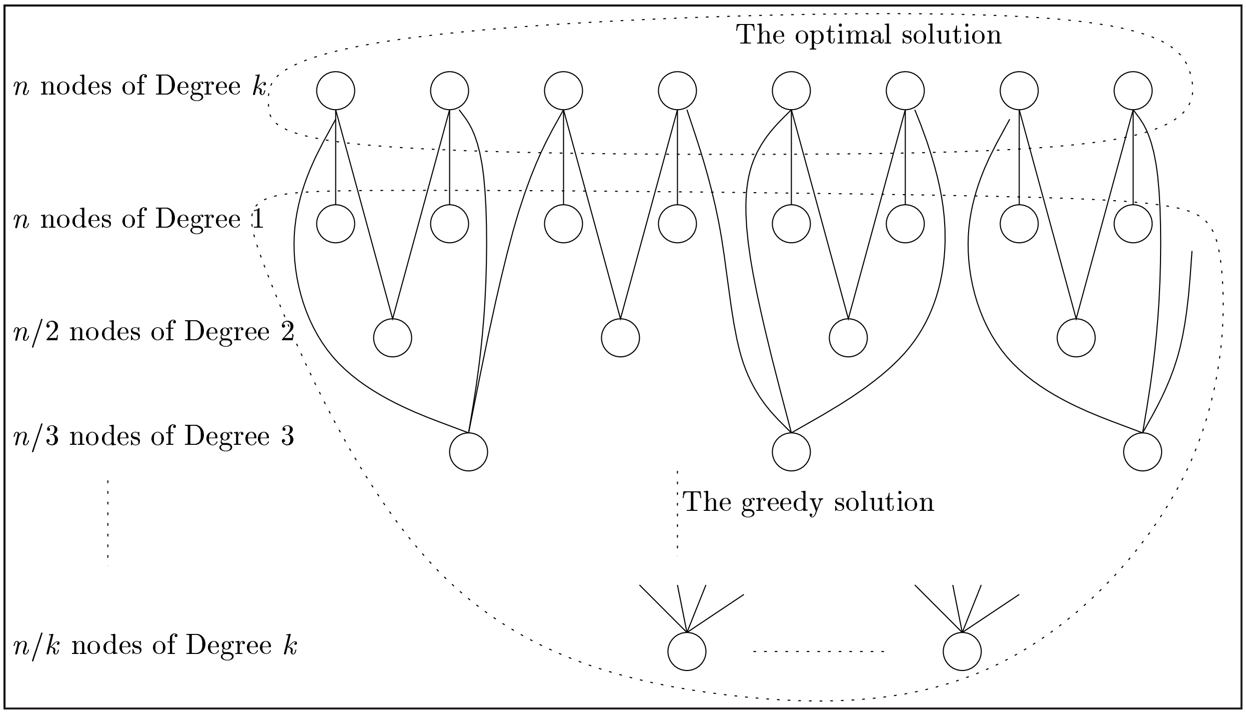

Figure 2.1: An example showing that the performance ratio of the greedy algorithm is not better than $H(\Delta)$. Here, the optimal solution is the first row (of size n), while the greedy algorithm might choose all the other rows, resulting in a set of size $nH(k)$.

We shall now describe several 2-approximation algorithms for the vertex cover problem. Each one of them uses a slightly different approach. It is an open problem whether there are α-approximation algorithms for vertex cover with $\alpha < 2$.

2.2.1 A simple approximation algorithm

The algorithm:

> Start with an empty cover $C \leftarrow \phi$.
> As long as there is an uncovered edge, add both its endpoints to the cover set C.

Correctness: The collection M of the edges whose endpoints were added to C is clearly a maximal matching (under containment). Every cover must include at least one endpoint of each ege of the matching M and therefore $|C| = 2|M| \leq 2|C^*|$. The algorithm achieves, therefore, a performance ratio of 2.

It is interesting to note that the above algorithm uses a greedy algorithm for constructing a *maximal* matching in G. A maximum matching of G can also be found in polynomial time. Here, however, we are interested in the opposite, we would like to find a maximal matching of minimal size. Unfortunately, this task is NP-hard [].

Questions:

1. Finding a minimum size maximal matching is NP-Hard. Can we find good approximation algorithms for the problem?
2. What is the preformance ratio achieved by the (non-polynomial) algorithm that finds a minimum size maximal matching and returns the cover composed of the endpoints of the edges in that matching? Is it still 2?

2.2.2 A simple randomized approximation algorithm (Pitt's Algorithm)

The algorithm:

> Start with an empty cover $C \leftarrow \phi$.
> As long as there is an uncovered edge, choose randomly one of the edge's endpoints and add it to the cover C.

Claim 2.1 $E(|C|) \leq 2|C^*|$

Proof: Let C^* be a minimum size cover. At least one endpoint of every edge examined by Pitt's algorithm belongs to C^*. With probability $\geq 1/2$ the algorithm adds to C a vertex that belongs to C^*. We consider such a choice to be a *successful* choice. The algorithm can make at most $|C^*|$ succesful choices, as after that $C^* \subseteq C$ and C is a cover. It follows, therefore, that the expected number of choices made by the algorithm, which is the size of C, is at most $2|C^*|$. ■

2.3 The weighted vertex cover problem

> Every vertex v of the graph $G = (V, E)$ now has a non-negative weight $w(v)$. The goal is to find a vertex cover C for which $w(C) = \sum_{v \in C} w(v)$ is minimal. This is clearly an extention of the vertex cover problem.

One can use Pitt's algorithm to obtain a 2-approximation for vertex cover using the following relatively simple modification: if an edge (u, v) is not covered yet, then add u to the cover with probability $\frac{1}{w(u)}/(\frac{1}{w(u)} + \frac{1}{w(v)})$, or v with the complementary probability $\frac{1}{w(v)}/(\frac{1}{w(u)} + \frac{1}{w(v)})$. (We state this result without proof).

We now describe several deterministic 2-approximation algorithms for weighted vertex cover. We start by considering the integer programming formulation of weighted vertex cover (IP):

$$\begin{aligned} &\min \sum_{v \in V} w(v)x(v) && \\ &x(v_1) + x(v_2) \geq 1 \quad , && (v_1, v_2) \in E \\ &x(v) \in \{0, 1\} \quad , && v \in V \end{aligned} \tag{2.4}$$

As before, we can solve an LP relaxation in which the constrains (2.4) are replaced by

$$x(v) \geq 0 \quad , v \in V \tag{2.5}$$

Given an optimal solution $\hat{x}(v)$ of the LP, we define an integer solution in the following way:

$$x(v) = \begin{cases} 1 & \text{if } \hat{x}(v) \geq 1/2 \\ 0 & \text{Otherwise} \end{cases}$$

and $C = \{v \in V \mid x(v) = 1\}$. This is a feasible solution (i.e., a cover) because for every $(v_1, v_2) \in E$ we have $\hat{x}(v_1) + \hat{x}(v_2) \geq 1$, so either $\hat{x}(v_1) \geq 1/2$ or $\hat{x}(v_2) \geq 1/2$. Furthermore,

$$w(C) = \sum_{v \in V} w(v)x(v) \leq 2 \sum_{v \in V} w(v)\hat{x}(v) \leq 2w(C^*) \ .$$

We thus obtain a 2-approximation algorithm for weighted vertex cover suggested by Hochbaum []. It is interesting to note that while the linear programming relaxation used here is similar to the linear programming relaxation of the hitting set cover, we are using this time a deterministic rounding procedure, and not a randomized one.

Can we avoid solving a linear program and still obtain a performance ratio of 2 for weighted vertex cover? The answer is yes. The following 2-approximation algorithm, due to Even and Bar-Yehuda [EvBa85] (see also [BarY97]) does not use linear programming and runs in linear time.

> As long as there is an edge $e = (v_1, v_2)$ for which $w(v_1) > 0$ and $w(v_2) > 0$, do
>
> $$\begin{gathered} \epsilon \leftarrow \min\{w(v_1), w(v_2)\} \\ w(v_1) = w(v_1) - \epsilon \\ w(v_2) = w(v_2) - \epsilon\} \end{gathered}$$
>
> Add all vertices whose weights are now 0 to the cover.

The set produced by the algorithm is clearly a vertex cover. We prove that its weight is at most twice the optimal using a *local ratio* argument. First, we present a recursive version of the algorithm:

> If for every edge $e = (v_1, v_2) \in E$, either $w(v_1) = 0$ or $w(v_2) = 0$ return the cover composed of all vertices of weight 0. This is obviously an optimal cover (in terms of weight).
>
> Otherwise, let $e = (v_1, v_2)$ be an edge for which $w(v_1) > 0$ and $w(v_2) > 0$. Define
>
> $$\delta(v) = \begin{cases} \min\{w(v_1), w(v_2)\} & \text{if } v \in \{v_1, v_2\} \\ 0 & \text{otherwise} \end{cases}$$
>
> It is easy to see that $0 \leq \delta(v) \leq w(v)$, for every $v \in V$. Find recursively a cover of G with the weight function $w - \delta$ and return it.

It is easy to verify that this algorithm works exactly like the original algorithm.

We denote by $\mathrm{OPT}(w)$, $\mathrm{OPT}(w - \delta)$ and $\mathrm{OPT}(\delta)$ the weights of the minimum covers in the graph G with the weight functions w, $w - \delta$ and δ. Let C^*, $C^*_{w-\delta}$ and C^*_δ be covers of minimum weight with respect to the corresponding weight functions. We have

$$w(C^*) = (w - \delta)(C^*) + \delta(C^*) \geq (w - \delta)(C^*_{w-\delta}) + \delta(C^*_\delta)$$

and thus

$$\mathrm{OPT}(w) \geq \mathrm{OPT}(w-\delta) + \mathrm{OPT}(\delta) \ .$$

The recursive algorithm described above clearly halts, as in each recrsive call the number of vertices with non-zero weight increases by at least 1. We prove, by induction, that the weight of the cover returned by the algorithm is at most twice the optimum. This clearly holds when the set of vertices with zero weight form a vertex cover, in this case the cover returned by the algorithm is of minimum weight.

Suppose now that the set of vertices with zero weight do not form a vertex cover. The algorithm finds, recursively, a cover of G with respect to the weight function $w-\delta$ and returns it. By the induction hypothesis $(w-\delta)(C) \leq 2\mathrm{OPT}(w-\delta)$. Also, we have that $\delta(C) \leq 2\min\{w(v_1), w(v_2)\} \leq 2\mathrm{OPT}(\delta)$ and therefore

$$w(C) \leq 2(\mathrm{OPT}(w-\delta) + \mathrm{OPT}(\delta)) \leq 2\mathrm{OPT}(w) \ ,$$

as required.

We present now another argument that shows that the above algorithm is a 2-approximation algorithm. We use this time the (weak) duality of linear programs.

Let (LP) be a linear program of the following form:

$$\begin{aligned} &\min_x c^T x \\ &Ax \geq b \\ &x \geq 0 \end{aligned}$$

The *dual* of the linear program (LP) is defined to be:

$$\begin{aligned} &\max_y b^T y \\ &A^T y \leq c \\ &y \geq 0 \end{aligned}$$

Theorem 2.2 (Duality of linear programs)

1. *For every x and y satisfying the relevant constrains, we have $c^T x \geq b^T y$.*
2. *If both the primal and dual linear programs have feasible solutions, then the optimal values of the two linear programs coincide.*

We will only use the first condition, sometimes referred to as *weak duality*, whose proof is immediate.

Proof: Simple manipulations show that

$$c^T x \geq (A^T y)^T x = y^T A x \geq y^T b = b^T y \ .$$

■

We return now to the weighted vertex cover problem. As we show, its LP relaxation is

$$\begin{aligned} &\min_x \sum_{v \in V} w(v)x(v) \\ &x(v_1) + x(v_2) \geq 1 \quad , \quad (v_1, v_2) \in E \\ &x(v) \geq 0 \qquad\qquad\quad , \quad v \in V \end{aligned}$$

The dual problem is

$$\begin{aligned} &\max_y \sum_{v \in V} y(v) \\ &\sum_{e:\ v \in e} y(e) \leq w(v) \quad , \quad v \in V \\ &y(e) \geq 0 \qquad\qquad\quad e \in E \end{aligned} \tag{2.6}$$

By weak duality, we have that $w(C^*) \geq \sum_{e \in E} y(e)$, for any feasible solution $y(e)$ of the dual. This suggests the following algorithm:

> Let $y(e) \leftarrow 0$ for every $e \in E$,
>
> As long as there is an edge e such that the dual constraints (constraints (2.6) at both its endpoints are not tight, increase $y(e)$ until at least one of these constraints becomes tight.
>
> Return a set cover C composed of all the vertices for which constraint 2.6 is tight, to the set C.

The set C is clearly a cover. Moreover

$$w(C) = \sum_{v \in C} w(v) = \sum_{v \in C} \sum_{e:\ v \in e} y(e) \leq 2 \sum_{e \in E} y(e) \leq 2w(C^*) \ .$$

We again obtain, therefore, a 2-approximation algorithm for weighted vertex cover. It is not difficult to see that algorithm obtained is again a reformulation of the algorithm of Bar-Yehuda and Even.

As mentioned, 2 is the the best relative performance guarantee available for vertex cover, even for the unweighted version. (Several algorithms achieve a ratio of $2 - \Omega(\frac{\log \log n}{\log n})$ but this is still 2, asymptotically).

Håstad [Ha97] showed recently that no polynomial time algorithm can give a ratio of less than 7/6, unless P=NP.

2.4 The Knapsack problem

The knapsack problem is:

> Given integral weights $w_1, \ldots, w_n$, integral values $v_1, \ldots, v_n$ and a budget W, find a subset S of $\{1, \ldots, n\}$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

The knapsack problem is known to be NP-hard. There is, however, a simple pseudo-polynomial algorithm for its solution:

Let $V = \max\{v_1, \ldots, v_n\}$. We assume $w_i \leq W$, for every $1 \leq i \leq n$ (if $w_i > W$ then the i-th item can never be used and it can be ignored). Let $w(i, v)$ denote the minimum weight needed to get the value v while using a subset of the first i items ($w(i, v) = \infty$ if this is not possible). We can compute the $w(i, v)$'s using dynamic programming:

$$\begin{aligned} w(0,0) &= 0 \\ w(0,v) &= \infty && , \quad v > 0 \\ w(i+1,v) &= w(i,v) && , \quad v < v_{i+1} \\ w(i+1,v) &= \min\{w(i,v), w(i, v - v_{i+1}) + w_{i+1}\} && , \quad v_{i+1} \leq v \leq nV \end{aligned}$$

The solution for the knapsack is the maximal v for which $w(n, v) \leq W$.

The complexity of this algorithm is $O(n \cdot nV) = O(n^2 V)$.

The pseudo-polynomial algorithm gives us immediately a Fully Polynomial Time Approximation Scheme (FPTAS) for knapscak.

Given $\epsilon > 0$. Define $b = \left\lfloor \log \frac{\epsilon V}{n} \right\rfloor$. We replace the b least significant bits of each value v_i by zeros, i.e., we let $v_i' \leftarrow 2^b \left\lfloor \frac{v_i}{2^b} \right\rfloor$, for $1 \leq i \leq n$. We now run the pseudo-polynomial time algorithm with the new values v_i' (or equivalently, with the values $v_i'/2^b$). The running time would be $O(n^2 V / 2^b) = O(n^3/\epsilon)$.

Let S^* be an optimal solution of the knapsack problem and let S be the solution returned by the algorithm. Then:

$$\mathrm{OPT} = \sum_{i \in S^*} v_i \geq \sum_{i \in S} v_i \geq \sum_{i \in S} v_i' \geq \sum_{i \in S^*} v_i' \geq \sum_{i \in S^*} (v_i - 2^b) \geq \left(\sum_{i \in S^*} v_i \right) - n2^b = \mathrm{OPT} - n2^b$$

Note that $\frac{n2^b}{\mathrm{OPT}} \leq \frac{n2^b}{V} \leq \frac{n}{V} \cdot \frac{\epsilon V}{n} = \epsilon$ and therefore $\sum_{i \in S} v_i \geq (1 - \epsilon)\mathrm{OPT}$.

2.5 Bin Packing

The bin packing problem is:

Given a collection of n items, where the i-th item is of size of s_i, where, find the minimum number of bins of size 1 into which all the elements can be fitted.

The bin packing problem is NP-Hard. Furthermore, even if we know in advance that all the elements can be packed into 3 bins, it is still NP-hard to decide whether they can be packed into 2 bins (this follows by reduction from the partition problem). Thus it is impossible to get an approximation ratio better than 3/2, unless P=NP. On the other side, as we shall see, for every $\epsilon > 0$ there is polynomial time algorithm that packs the items of an instance I into $(1+\epsilon)\mathrm{OPT}(I) + 1$ bins, where $\mathrm{OPT}(I)$ is of course the minimum number of bins needed. We say that such an algorithm achieves an *asymptotic approximation ratio* of $1 + \epsilon$. The collection of these algorithms is said to be an *Asymtotic Polynomial Time Approximation Approximation Scheme* (APTAS).

The APTAS for bin packing will be described in the next lecture. We finish the lecture today by considering a very simple heuristic for bin packing – first fit.

2.5.1 The first fit heuristic

> Process the the items sequentially. Put each item in the first bin into which it fits. If there is no such bin then put the item in a new bin.

Claim 2.3 *For every instance I we have $FF(I) \leq 2OPT(I) + 1$.*

Proof: At the end of the process, there is at most one bin that is more than half empty, otherwise the items in the second such bin could have been packed into the first such bin. Thus:

$$\mathrm{SIZE}(I) = \sum_{i=1}^{n} s_i \geq \tfrac{1}{2}(\mathrm{FF}(I) - 1)\ .$$

$$\mathrm{FF}(I) \leq 2 \cdot \mathrm{SIZE}(I) + 1 \leq 2 \cdot \mathrm{OPT}(I) + 1\ .$$

■

It is known that the asymptotic approximation ratio of First Fit is exactly 1.7. The proof, however, is very lengthy []. It is relatively easy to show lower bound of 5/3 on the performance ratio of First Fit:

Let I_m be the following instance:

1. m items of size $\frac{1}{7} + \epsilon$.
2. m items of size $\frac{1}{3} + \epsilon$.
3. m items of size $\frac{1}{2} + \epsilon$.

Clearly $\mathrm{OPT}(I_m) \leq m$, but $\mathrm{FF}(I_m) = \frac{m}{6} + \frac{m}{2} + m = \frac{5}{3}m$.