An Approximation Algorithm for the Cutting-Sticks Problem

Jagadish M[1]

Indian Institute of Technology Bombay
Powai
Mumbai 400076

Abstract

The *cuttings-sticks* problem is the following. We are given a bundle of sticks all having integer lengths. The total sum of their lengths is $n(n+1)/2$. Can we break the sticks so that the resulting sticks have lengths $1, 2, \ldots, n$? The problem is known to be NP-hard. We consider an optimization version of the problem which involves cutting the sticks and placing them into boxes. The problem has a trivial polynomial time algorithm with an approximation ratio of 2. We present a greedy algorithm that achieves an approximation ratio of $\sqrt{2}$.

Keywords: Approximation algorithms, Cutting Sticks

---

1. Introduction

Cutting-sticks problem. We are given k sticks all having integer lengths. Their lengths are $l_1, \ldots, l_k$ and the total sum of their lengths is $n(n+1)/2$. Can we break the sticks to get sticks of lengths $1, 2, \ldots, n$?

Notation. Let $[r]$ denote the set $\{1, 2, \ldots, r\}$.

We propose an optimization version of the cutting-sticks problem and give an approximation algorithm for it. We first state the decision version of the problem slightly differently to resemble the optimization version.

Decision version. We are given k positive integers $l_1, \ldots, l_k$ as input. Their total sum is $n(n+1)/2$. Can we partition the set $[n]$ into k subsets $B_1, \ldots, B_k$ such that the sum of numbers in B_i equals l_i for all $1 \leq i \leq k$?

---

[1]Email: `jagadish@cse.iitb.ac.in`

It is easy to see that the alternate formulation does not change the problem. For any positive instance, the ith stick can be broken down into sizes contained in the set B_i.

Optimization version. Given positive integers $l_1, \ldots, l_k$ as input, find the smallest t for which we can partition the set $[t]$ into k subsets $B_1, \ldots, B_k$ such that the sum of numbers in B_i is at least l_i for all $1 \leq i \leq k$. Output t and the corresponding partition in polynomial time.

Related work. The cutting-sticks problem is NP-hard since the 3-partition problem reduces to it Ito (2010). For the special case when all the lengths are equal, there exists a polynomial time algorithm to solve the problem exactly Straight and Schillo (1979). We give a $\sqrt{2}$-approximation factor algorithm for the optimization version of the problem. Suppose we are given an instance of the problem that has $t = OPT$ as the output. Our algorithm outputs a number that is at most $\sqrt{2}OPT$ and gives the corresponding partitioning of the set $\left[\lceil\sqrt{2}OPT\rceil\right]$.

1.1. Physical Interpretation

For ease of exposition, we give a physical interpretation of the optimization problem. Roughly, we can see the problem as breaking a set of sticks in order to fit them into a set of boxes.

Definition. The *box* b_i is of *size* i. Box b_i can *contain* a stick of length j if $j \leq i$. Equivalently, a stick of length j can *fit* into a box whose size is at least j. We refer to boxes $b_1, \ldots, b_t$ as '$[t]$ boxes'.

The set of sticks is said to *fit into* $[t]$ *boxes*, if the sticks can be broken into shorter sticks (*pieces*) such that:

1. Each piece fits into some box b_i.
2. One box contains at most one piece. (Some boxes could remain empty.)

In light of the above definitions, the optimization problem we want to solve can be stated as follows. We are given k sticks $s_1 \ldots, s_k$ having lengths $l_1, \ldots, l_k$, respectively. What is the smallest number t for which we can we fit the sticks into $[t]$ boxes?

For example, if the stick lengths are 6, 5 and 4, then smallest t for which we can fit the sticks into $[t]$ boxes is 5 (See Fig. 1).

If we can solve the above problem exactly, we can solve the decision version of the cutting-sticks problem too. Since the latter problem is NP-hard, we will look for an approximation algorithm. We give a polynomial time algorithm with approximation factor $\sqrt{2}$ for the above problem. In other words, suppose OPT is the smallest t for which we can fit the given sticks

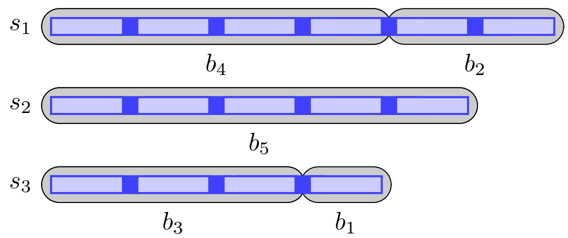

Figure 1: An optimal way to fit sticks of lengths 6, 5 and 4 into [5] boxes. The first stick is broken into two pieces of lengths 4 and 2. The second stick is a piece by itself. The third stick is broken into pieces of lengths 3 and 1. The corresponding solution to the original optimization problem is $B_1 = \{4, 2\}, B_2 = \{5\}$ and $B_3 = \{3, 1\}$. If the sticks were of lengths 6, 5 and 3, an optimal solution would still use [5] boxes and could have the same partition (box b_1 would be left empty).

into $[t]$ boxes. Our algorithm outputs a number that is at most $\sqrt{2}OPT$ and gives a way to fit the sticks into $[\sqrt{2}OPT]$ boxes.

2. Greedy Algorithm

Assumption. We make a simplifying assumption that we know the value of OPT prior to the start of the algorithm. This assumption can be removed by binary searching for the minimum value for which the algorithm returns a solution. We defer the details to Section 4.5.

Informally, our algorithm works as follows. At each step, we pick the longest stick and either cut from one end of stick and place the piece into the largest empty box or place the stick itself into the box. The exact algorithm is given below. The parameter α denotes the approximation factor of the algorithm whose value will be fixed to $\sqrt{2}$ with hindsight.

Initially, we have $[\alpha OPT]$ empty boxes and the sticks labelled $s_1, \ldots, s_k$.

1. Pick the largest empty box available (say b_i). Note that box b_i has size i. Pick the stick with the longest remaining length (say s_j). Let $len(s_j)$ denote the current length of s_j.
   (a) If $len(s_j) \leq i$, then fit the remaining portion of the stick s_j into b_i.

(b) Otherwise, cut the stick s_j from one end to get a piece of length i and place this piece into the box b_i. The stick s_j now has length equal to $len(s_j) - i$.

In either case, box b_i becomes non-empty after this step.

2. Repeat Step 1 until no empty box is available or until all the sticks are fit into boxes.

Fig. 2 shows the run of the algorithm on the input $l_1 = 6, l_2 = 5, l_3 = 4$ with $OPT = 5$. The example shows a bad case when the algorithm fails when run with $\alpha = 1$. The algorithm considers boxes in decreasing order of their sizes. In the first step, box b_5 is picked and a portion equal to length 5 is cut from the stick s_1. The remaining portion of the stick s_1 has length 1. In the second step, the box b_4 is picked and a piece of size 4 is cut from s_2 and placed in b_4. In the next step b_3 is picked and so on. The greedy algorithm fails to fit all the sticks if it starts with $\alpha = 1$. In the next section, we prove that if $\alpha \geq \sqrt{2}$, then the algorithm succeeds for any input.

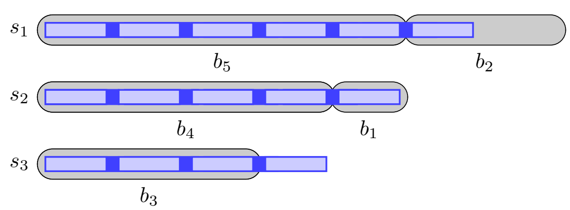

Figure 2: An execution of the greedy algorithm with [5] boxes. Although the sticks fit into [5] boxes optimally, the greedy algorithm fails to do so: a portion of the stick s_3 is not fit into any box. It can be easily checked that if the greedy algorithm started with [6] boxes, then it would succeed in fitting all the sticks.

3. A lower bound on *OPT*

We are given an input with the guarantee that all the sticks will fit into $[OPT]$ boxes. To prove an approximation ratio, we first need a lower bound on the value of OPT. We express lower bounds on OPT in terms of two quantities of the input: number of sticks and the total length of the sticks.

Claim 3.1. *The number of sticks is at most OPT.*

Proof. Every stick requires at least one box and there are only OPT boxes. $\square$

Notation. Let $\langle r \rangle$ denote the quantity $r(r+1)/2$.

Claim 3.2. *The total length of all the sticks is at most* $\langle OPT \rangle$.

Proof. All the sticks fit into $[OPT]$ blocks. The total size of boxes is $\langle OPT \rangle$. $\square$

4. Analysis

We prove that the above algorithm has an approximation ratio of $\sqrt{2}$. We first discuss a weaker result that brings out a few aspects of our actual analysis.

Definitions. A *partially filled box* is one which contains a piece of stick that is *strictly* smaller than its size (the box's size). Box b_i is a partially-filled box if it contains a piece of length strictly smaller than i. A *completely filled box* is one that contains a piece exactly equal to its size. For example, in Fig. 2 b_2 is a partially-filled box, while the rest of them are completely-filled boxes. In Fig. 1, all the boxes are completely filled. We refer to completely-filled boxes as c-boxes and partially filled boxes as p-boxes. A box which has no stick after the termination of the algorithm is called an *empty* box.

4.1. 2-approximation factor

Claim 4.1. *The greedy algorithm succeeds in fitting all the sticks if it starts with* $[2OPT]$ *blocks.*

Proof. Assume, for contradiction's sake, that the algorithm has terminated without fitting all the sticks. The greedy algorithm, when run, induces a labelling on the boxes (Fig. 3). Each box is either a c-box or a p-box. Since the algorithm has not fitted all the sticks, it has run out of boxes. This means each box of the $[2OPT]$ boxes must be either completely filled or partially filled. Since there are $2OPT$ boxes, either the number of c-boxes or p-boxes is at least equal to OPT. Each c-box contains a piece equal to its size. so we cannot have more than OPT c-blocks, since the sum of any set of OPT numbers from $[2OPT]$ is at least $\langle OPT \rangle$ (refutes Claim 3.2). There cannot be more than OPT p-blocks since that would mean there are more than OPT sticks (refutes Claim 3.1). $\square$

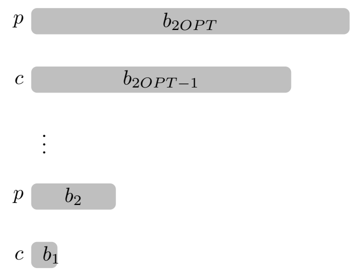

Figure 3: The greedy algorithm, when run on an input, induces a labelling on the boxes. Each box is either a c-box or a p-box

4.2. $\sqrt{2}$-approximation factor

We make an observation that lets us improve the approximation bound substantially. Recall that our algorithm picks the boxes in decreasing order of their size. We try to fit the stick with the longest remaining length into the current box.

Key Fact. Suppose b_x is a partially filled box and b_y is a completely filled box, with $x > y$. Since a stick of length y was fitted into b_y, at least a stick of length y must have been available to box b_x since it was picked before b_y. Therefore, b_x must contain a stick of length at least y units.

Here is an intuitive reason why this fact is helpful. In the proof of Claim 4.1, we did not take into account the stick lengths contained in the partially-filled boxes. The above observation helps us to incorporate the stick lengths in *all* the boxes into the analysis. We introduce a quantity called *cover* that will be useful in bounding the total length of sticks.

4.2.1. Definitions

Labelling. An assignment of each box in $[m]$ to either 'p' or 'c' is called a *labelling*. Boxes with label p are called p-boxes and those with label c are called c-boxes (with slight abuse of terminology).

Cover. Each box in $[m]$ is associated with a natural number called *cover*. Given a labelling of $[m]$, the cover of a c-box is equal to its size and the cover of a p-box is equal to the size of the largest c-box less than its size.

Total cover. The *total cover* of a labelling is the sum of all the covers of the boxes. Note that total cover is a property of the labelling. The total cover of a labelling $\mathcal{X}$ is denoted by $t_c(\mathcal{X})$.

For example, a labelling for [8] boxes is shown in Fig. 4. The cover of each box is in dark shade. The c-box b_7 has cover of 4, since b_4 is the largest c-box less than b_7. Likewise, boxes b_3 and b_2 have covers of 1 each since b_1 is a c-box. The total cover of the labelling is 27 $(8\times1+4\times4+1\times3)$. It is easy to see that the cover of each box and subsequently the total cover is uniquely determined by the labelling.

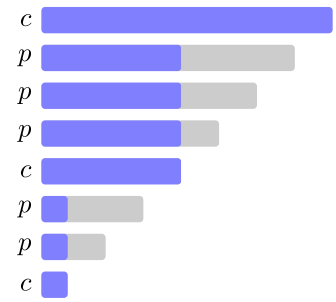

Figure 4: A labelling and corresponding covers (in blue).

4.3. Relating the total cover to the greedy algorithm

Suppose there exists a bad instance on which the greedy algorithm fails when run with $[\alpha OPT]$ boxes, for some $\alpha > \sqrt{2}$. Let $\mathcal{L}$ be the labelling of the boxes that results from running the algorithm on this instance. This means $\mathcal{L}$ is a labelling defined on $[\alpha OPT]$ boxes in which partially filled boxes at the end of the algorithm are labelled 'p' and completely filled boxes are labelled 'c'.

The following lemma relates the total stick lengths contained in the boxes to the value of $t_c(\mathcal{L})$.

Lemma 4.2. *The value of the total stick lengths contained in the boxes at the end of the algorithm is at least* $t_c(\mathcal{L})$.

Proof. We show that the value of the stick length contained in each box is at least the cover of that box. Let us consider a box b_i. There are only two cases:

- If b_i is a c-box, then the stick length contained in it is equal to i. This is the value of the cover of box b_i.
- If b_i is a p-box, then the stick length contained in it is at least as large as the value of the cover of box b_i (follows from the key fact and our definition of cover).

□

Notation. For a labelling $\mathcal{X}$, let $n_p(\mathcal{X})$ denote the number of p-boxes in the labelling $\mathcal{X}$.

The labelling induced by the algorithm must satisfy the following two conditions:

Condition 1. $n_p(\mathcal{L}) \leq OPT$. The number of p-boxes should be at most the number of sticks in the input. Since the number of sticks at most OPT (Claim 3.1), the number of p-boxes should be at most OPT.

Condition 2. $t_c(\mathcal{L}) \leq \langle OPT \rangle$. The value of the total length of the sticks contained in the boxes is at least as much as the total cover of $\mathcal{L}$ (Lemma 4.2). We know that the former quantity is at most $\langle OPT \rangle$ (Claim 3.2). Hence, the total cover of $\mathcal{L}$ should be smaller than $\langle OPT \rangle$.

In the next section, we prove that if $\alpha > \sqrt{2}$, then any labelling $\mathcal{L}$ will violate one of the two conditions. This means that no bad instance exists for the greedy algorithm if it starts with $\lceil \sqrt{2} OPT \rceil$ boxes. This implies that the approximation ratio of the algorithm is $\sqrt{2}$.

Claim 4.3. *The greedy algorithm has approximation ratio of $\sqrt{2}$.*

Proof. Assume, for a contradiction, that there exists a bad instance on which the greedy algorithm fails when run with $\alpha > \sqrt{2}$. The induced labelling $\mathcal{L}$ must satisfy the following constraints:

$$t_c(\mathcal{L}) \leq \langle OPT \rangle \qquad (Condition\ 2)$$

In Section 4.4, we prove the following result (Lemma 4.7).

$$t_c(\mathcal{L}) \geq \langle (\alpha - 1) OPT \rangle + (\alpha - 1) OPT^2$$

Hence,

$$\langle (\alpha - 1) OPT \rangle + (\alpha - 1) OPT^2 \leq \langle OPT \rangle$$

Ignoring lower order terms:

$$\frac{(\alpha - 1)^2 OPT^2}{2} + (\alpha - 1) OPT^2 \leq \frac{OPT^2}{2}$$

$$\alpha \leq \sqrt{2} \text{ (contradiction)}$$

□

4.4. The minimum value of the total cover of $\mathcal{L}$

In this section, we lower bound the value of the total cover of the labelling $\mathcal{L}$. We do so by defining another intermediate labelling $\mathcal{L}''$ whose total cover is easy to calculate and whose total cover is not more than $t_c(\mathcal{L})$.

An exchange argument. Let b_i be the smallest p-box and b_j be the smallest c-box in $\mathcal{L}$. Suppose $j > i$. Let $\mathcal{L}'$ be the labelling obtained from $\mathcal{L}$ by swapping the labels of b_i and b_j. In the example shown in Fig. 5, $i = 2$ and $j = 4$.

Lemma 4.4. $t_c(\mathcal{L}) \geq t_c(\mathcal{L}')$.

Proof. Let us see how the swapping of labels affects the cover of each box in $\mathcal{L}'$ compared to the corresponding box in $\mathcal{L}$.

- The cover of boxes larger than b_j can decrease or remain the same.
- The cover of b_j decreases by $j - i$ (box b_j as a covering box had cover of j before, after the relabelling it has cover of i).
- The cover of boxes b_{j-1} through b_i increase by 1 (every box had a cover of $i - 1$ before relabelling and i after). This results in a total increase of $j - i$.
- The cover of boxes smaller than b_i remain the same.

Hence the total cover of $\mathcal{L}'$ is less than or equal to that of $\mathcal{L}$. □

Let $\mathcal{L}''$ be another labelling which has same number of p-boxes (and consequently c-boxes) as $\mathcal{L}$. In the labelling $\mathcal{L}''$ all the p-boxes appear before the c-boxes (Fig. 5). In other words, if b_u is a p-box and b_v is a c-box then it implies that $u > v$.

Lemma 4.5. $t_c(\mathcal{L}) \geq t_c(\mathcal{L}'')$.

Proof. If the labelling $\mathcal{L}'' \neq \mathcal{L}$, then $\mathcal{L}''$ can be obtained by repeatedly swapping the smallest p box and the smallest c box in $\mathcal{L}$. By the same argument as given in Lemma 4.4 the total cover can only decrease after each swap. □

Lemma 4.6. $t_c(\mathcal{L}'') \geq \langle(\alpha - 1)OPT\rangle + (\alpha - 1)OPT^2$.

Proof. We know that $n_p(\mathcal{L}'') = n_p(\mathcal{L})$ and $n_p(\mathcal{L}) \leq OPT$. Hence $\mathcal{L}''$ has at most OPT p-boxes and at least $(\alpha - 1)OPT$ c-boxes. Let us bound the total node cover of $\mathcal{L}''$. Every box b_i where $i \in \{1, \ldots, (\alpha - 1)OPT\}$ is a c-box so the sum of covers of those boxes is $\langle(\alpha - 1)OPT\rangle$. Every box b_i with $i > (\alpha - 1)OPT$ is either a c-box with cover i or an p-box with cover at least $(\alpha - 1)OPT$. So every box b_i with $i > (\alpha - 1)OPT$ has cover of at least $(\alpha - 1)OPT$. There are OPT boxes in the set $\{b_{(\alpha-1)OPT}, \ldots, b_{\alpha OPT}\}$ and their covers add up to at least $(\alpha - 1)OPT^2$. Therefore, $t_c(\mathcal{L}'') \geq \langle(\alpha - 1)OPT\rangle + (\alpha - 1)OPT^2$. □

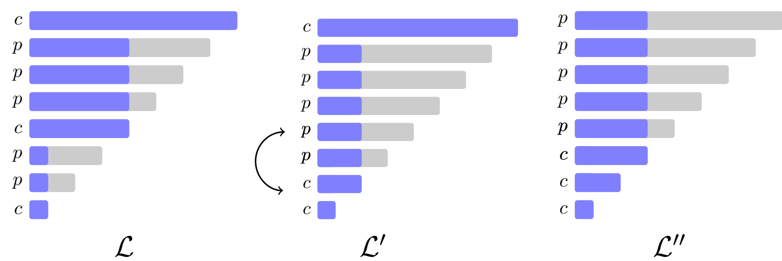

Figure 5: An example of a swapping operation. The labellings $\mathcal{L}'$ and $\mathcal{L}''$ each have smaller total cover than that of labelling $\mathcal{L}$.

Lemma 4.7.

$$t_c(\mathcal{L}) \geq \langle(\alpha - 1)OPT\rangle + (\alpha - 1)OPT^2$$

Proof. Follows from lemmas 4.5 and 4.6. □

4.5. Removing the assumption about knowing OPT

We made the simplifying assumption that we know the value of OPT. This can be removed as follows: We run the greedy algorithm with $[x]$ boxes for every value of $x \in 1, 2, 3, \ldots$ until the algorithm succeeds in fitting the sticks. This way we can find the smallest value of x for which the algorithm succeeds with $[x]$ boxes. Since the algorithm succeeds with $[\sqrt{2}OPT]$ boxes, x is guaranteed to be at most $\sqrt{2}OPT$. It is easy to see that the algorithm runs in time polynomial in OPT.

4.6. Lower bound

Claim 4.8. *The greedy algorithm cannot give better than* 1.16 *approximation.*

Proof. A bad instance for the greedy algorithm is the union of three sets of sticks: $S_1 \cup S_2 \cup S_3$, where

$$\begin{aligned} S_1 &= \ \{k/6 \text{ sticks of length } k\} \\ S_2 &= \{k, k-1, \ldots, 2k/3 + 1\} \\ S_3 &= \{k/3 - 1, k/3 - 2, \ldots, 1\} \end{aligned}$$

First, we prove that $OPT = k$ by showing that $S_1 \cup S_2 \cup S_3$ can fit into $[k]$ boxes. Set S_1 can fit into boxes of sizes $\{2k/3, \ldots, k/3\}$: Pick two boxes of size $k/2 + i$ and $k/2 - i$ for $1 \leq i \leq k/6$ for each stick in S_1. Each stick in S_1 will be cut into two equal pieces.

The set $S_2 \cup S_3$ can be covered by boxes of sizes from the set $[k] \setminus \{2k/3, \ldots, k/3\}$. Hence, $OPT = k$ for this input.

It is easy to verify that the greedy algorithm fails if fewer than $[7/6k - 2]$ boxes are taken. The problem arises due to sticks in S_2. Each box that the greedy algorithm picks for a stick in S_2 for the first time is only slightly smaller than the stick. For example, box of size $k - 2$ is picked when fitting a stick of k, box of size $k - 3$ is picked for a fitting a stick of size $k - 1$ and so on. This forces the algorithm to use two boxes to fit sticks in S_2. If fewer than $[7/6k - 2]$ boxes are taken, some piece of stick in S_2 is left out. $\square$

Acknowledgement. I thank Prof. Abhiram Ranade for pointing me to a reference on cutting-sticks problem.

References

Ito, T., 2010. Cutting-sticks puzzle. Theoretical Computer Science Stack Exchange, http://cstheory.stackexchange.com/q/713 (version: 2010-08-30).

Straight, H. J., Schillo, P., 1979. On the problem of partitioning $\{1, 2, \ldots, n\}$ into subsets having equal sums. Proceedings of the American Mathematical Society 74 (2), 229–231.