Cake-Cutting is Not a Piece of Cake

Malik Magdon-Ismail, Costas Busch, and Mukkai S. Krishnamoorthy

Department of Computer Science
Rensselaer Polytechnic Institute
110 8th Street, Troy, NY 12180
{magdon, buschc, moorthy}@cs.rpi.edu

Abstract. Fair cake-cutting is the division of a cake or resource among N users so that each user is content. Users may value a given piece of cake differently, and information about how a user values different parts of the cake can only be obtained by requesting users to "cut" pieces of the cake into specified ratios. One of the most interesting open questions is to determine the minimum number of cuts required to divide the cake fairly. It is known that $O(N \log N)$ cuts suffices, however, it is not known whether one can do better.

We show that sorting can be reduced to cake-cutting: *any* algorithm that performs fair cake-division can sort. For a general class of cake-cutting algorithms, which we call linearly-labeled, we obtain an $\Omega(N \log N)$ lower bound on their computational complexity. All the known cake-cutting algorithms fit into this general class, which leads us to conjecture that every cake-cutting algorithm is linearly-labeled. If in addition, the number of comparisons per cut is bounded (comparison-bounded algorithms), then we obtain an $\Omega(N \log N)$ lower bound on the number of cuts. All known algorithms are comparison-bounded.

We also study variations of envy-free cake-division, where each user feels that they have more cake than every other user. We construct utility functions for which any algorithm (including continuous algorithms) requires $\Omega(N^2)$ cuts to produce such divisions. These are the the first known general lower bounds for envy-free algorithms. Finally, we study another general class of algorithms called *phased* algorithms, for which we show that even if one is to simply guarantee each user a piece of cake with positive value, then $\Omega(N \log N)$ cuts are needed in the worst case. Many of the existing cake-cutting algorithms are phased.

1 Introduction

Property sharing problems, such as chore division, inheritance allocation, and room selection, have been extensively studied in economics and game-theory [1, 5, 7, 9, 11]. Property sharing problems arise often in everyday computing when different users compete for the same resources. Typical examples of such problems are: job scheduling; sharing the CPU time of a multiprocessor machine; sharing the bandwidth of a network connection; etc. The resource to be shared can be viewed as a cake, and the problem of sharing such a resource is called *cake-cutting* or *cake-division.* In this work, we study fair cake-division; we are interested in quantifying how hard a computational problem this is.

In the original formulation of the cake-division problem, introduced in the 1940's by Steinhaus [15], N users wish to share a cake in such a way that each user gets a portion of the cake that she is content with (a number of definitions of content can be considered, and we will discuss these more formally later). Users may value a given piece of the cake differently. For example, some users may prefer the part of the cake with the chocolate topping, and others may prefer the part without the topping. Suppose there are five users, and lets consider the situation from the first user's point of view. The result of a cake-division is that every user gets a portion of the cake, in particular user 1 gets a portion. User 1 will certainly not be content if the portion that she gets is worth (in her opinion) less than one fifth the value (in her opinion) of the entire cake. So in order to make user 1 content, we must give her a portion that she considers to be worth *at least* one fifth the value of the cake, and similarly for the other users. If we succeed in finding such a division for which all the users are content, we say that it is a *fair cake-division.*

More formally, we represent the cake as an interval $I = [0, 1]$. A piece of the cake corresponds to some sub-interval of this interval, and a portion of cake can be viewed as a collection of pieces. The user only knows how to value pieces as specified by her utility function, and has no knowledge about the utility functions of the other users. The cake-division process (an assignment of portions to users) is to be effected by a superuser $\mathcal{S}$ who initially has no knowledge about the utility functions of the users. We point out that this may appear to diverge from the accepted mentality in the field, where protocols are viewed as self-enforcing – players are advised as to how they should cut, and are guaranteed that if they cut according to the advice, then the result will be equitable to them. The way such algorithms usually proceed is that the players make cuts, and somehow based on these cuts, portions are assigned to the players. Some computing body needs to do this assignment, and perform the necessary calculations so that the resulting assignment is guaranteed to be equitable to all the players (provided that they followed the advice). It is exactly this computing body that we envision as the superuser, because we would ultimately like to quantify all the computation that takes place in the cake division process. In order to construct an appropriate division, the superuser may request users to cut pieces into ratios that the superuser may specify. Based on the information learned from a number of such cuts, the superuser must now make an appropriate assignment of portions, such that each user is content. A simple example will illustrate the process. Suppose that two users wish to share the cake. The superuser can ask one of the users to cut the entire cake into two equal parts. The superuser now asks the other user to evaluate the two resulting parts. The second user is then assigned the part that she had higher value for, and the first user gets the remaining part. This well known division scheme, sometimes termed "I cut, you choose", clearly leaves both users believing they have at least half the cake, and it is thus a successful fair division algorithm. From this example we see that one cut suffices to perform fair division for two users. An interesting question to ask is: what is the minimum number of cuts required to perform a fair division when there are N users.

The cake-division problem has been extensively studied in the literature [4, 6, 8, 12–14, 16, 17]. From a computational point of view, we want to minimize the number of cuts needed, since this leads to a smaller number of computational steps performed by the algorithm. Most of the algorithms proposed in the literature require $O(N^2)$ cuts for N users (see for example Algorithm A, Section 2.3), while the best known cake-cutting algorithm, which is based on a divide-and-conquer procedure, uses $O(N \log N)$ cuts (see for example Algorithm B, Section 2.3). More examples can be found in [4, 14]. It is not known whether one can do better than $O(N \log N)$. In fact, it is conjectured that there is no algorithm that uses $o(N \log N)$ cuts in the worst case. The problem of determining what the minimum number of cuts required to guarantee a fair cake-division seems to be a very hard one. We quote from Robertson and Webb [14, Chapter 2.7]:

> "The problem of determining in general the fewest number of cuts required for fair division seems to be a very hard one. ... We have lost bets before, but if we were asked to gaze into the crystal ball, we would place our money against finding a substantial improvement on the $N \log N$ bound."

Of course, a well known continuous protocol that uses $N - 1$ cuts is the moving knife algorithm (see Section 2.3). Certainly $N-1$ cuts cannot be beaten, however, such continuous algorithms are excluded from our present discussion. Our results apply only to discrete protocols except when explicitly stated otherwise.

Our main result is that sorting can be reduced to cake-cutting: *any* fair cake-cutting algorithm can be converted to an equivalent one that can sort an arbitrary sequence of distinct positive integers. Further, this new algorithm uses no more cuts (for any set of N users) than the original one did. Therefore, cake-cutting should be at least as hard as sorting. The heart of this reduction lies in a mechanism for labeling the pieces of the cake. Continuing, we define the class of linearly-labeled cake-cutting algorithms as those for which the extra cost of labeling is linear in the number of cuts. Essentially, the converted algorithm is as efficient as the original one. For the class of linearly-labeled algorithms, we obtain an $\Omega(N \log N)$ lower bound on their computational complexity. To our knowledge, all the known fair cake-cutting algorithms fit into this general class, which leads us to conjecture that every fair cake-cutting algorithm is linearly-labeled, a conjecture that we have not yet settled. From the practical point of view, the computational power of the superuser can be a limitation, and so we introduce the class of algorithms that allow the super user a budget, in terms of computation, for every cut that is made. Thus, the computation that the superuser performs can only grow linearly with the number of cuts performed. Such algorithms we term comparison-bounded. All the known algorithms are comparison-bounded. If in addition to being linearly-labeled, the algorithm is also comparison-bounded, then we obtain an $\Omega(N \log N)$ lower bound on the number of cuts required in the worst case. Thus the conjecture of Robertson and Webb is true within the class of linearly-labeled & comparison-bounded algorithms. To our knowledge, this class includes all the known algorithms, which makes it a very interesting and natural class of cake-cutting algorithms. These are the first "hardness" results for a general class of cake-cutting algorithms that are applicable to a general number of users.

We also provide lower bounds for some types of *envy-free* cake-division. A cake-division is envy-free if each user believes she has at least as large a portion (in her opinion) as every other user, i.e., no user is envious of another user. The two person fair division scheme presented earlier is also an envy-free division scheme. Other envy-free algorithms for more users can be found in [3, 4, 8, 11, 14]. It is known that for any set of utility functions there are envy-free solutions [14]. However, there are only a few envy-free algorithms known in the literature for N users [14]. Remarkably, all these algorithms are *unbounded*, in the sense that there exist utility functions for which the number of cuts is finite, but arbitrarily large. Again, no lower bounds on the number cuts required for envy-free division exist. We give the first such lower bounds for two variations of the envy-free problem, and our bounds are applicable to both discrete and continuous algorithms. A division is *strong envy-free*, if each user believes she has *more* cake than the other users, i.e., each user believes the other users will be envious of her. We show that $\Omega(N^2)$ cuts are required in the worst case to guarantee a strong envy-free division when it exists. A division is *super envy-free*, if every user believes that every other user has at most a fair share of the cake (see for example [2, 14]. We show that $\Omega(N^2)$ cuts are required in the worst case to guarantee super envy-free division when it exists. These lower bounds give a first explanation of why the problem of envy-free cake-division is harder than fair cake-division for general N.

The last class of cake-cutting algorithms that we consider are called *phased* algorithms. In phased algorithms, the execution of the algorithm is partitioned into phases. At each phase all the "active" users make a cut. At the end of a phase users may be assigned portions, in which case they become "inactive" for the remainder of the algorithm. Many known cake-cutting algorithms are phased. We show that there are utility functions for which *any* phased cake-cutting algorithm requires $\Omega(N \log N)$ cuts to guarantee every user a portion they believe to be of positive value (a much weaker condition than fair). For such algorithms, assigning positive portions alone is hard, so requiring the portions to also be fair or envy-free can only make the problem harder. In particular, Algorithm B (see Section 2.3), a well known divide and conquer algorithm is phased, and obtains a fair division using $O(N \log N)$ cuts. Therefore this algorithm is optimal among the class of phased algorithms *even* if we compare to algorithms that merely assign positive value portions. The issue of determining the maximum value that can be guaranteed to every user with K cuts has been studied in the literature [14, Chapter 9]. We have that for phased algorithms, this maximum value is zero for $K = o(N \log N)$.

The outline of the remainder of the paper is as follows. In the next section, we present the formal definitions of the cake-division model that we use, and what constitutes a cake-cutting algorithm, followed by some example algorithms. Next, we present the lower bounds. In Section 3 we introduce phased algorithms and give lower bounds for the number of cuts needed. Section 4 discusses labeled algorithms and the connection to sorting. In Section 5 we give the lower bounds for envy-free division, and finally, we make some concluding remarks in Section 6. Due to space constraints, we refer to the accompanying technical report [10] for most of the proofs.

2 Preliminaries

2.1 Cake-Division

We denote the cake as the interval $I = [0, 1]$. A *piece* of the cake is any interval $P = [l, r]$, $0 \leq l \leq r \leq 1$, where l is the left end point and r the right end point of P. The *width* of P is $r - l$, and we take the width of the empty set $\emptyset$ to be zero. $P_1 = [l_1, r_1]$ and $P_2 = [l_2, r_2]$ are *separated* if the width of $P_1 \cap P_2$ is 0, otherwise we say that P_1 and P_2 *overlap*. P_1 *contains* P_2 if $P_2 \subseteq P_1$. If P_1 and P_2 are separated, then we say that P_1 is left of P_2 if $l_1 < l_2$. The *concatenation* of the $M > 1$ pieces $\{[l, s_1], [s_1, s_2], [s_2, s_3], \ldots, [s_{M-1}, r]\}$ is the piece $[l, r]$.

A *portion* is a non-empty set of separated pieces $\mathcal{W} = \{P_1, P_2, \ldots, P_k\}$, $k \geq 1$. Note that a portion may consist of pieces which are not adjacent (i.e. a portion might be a collection of "crumbs" from different parts of the cake). Two portions $\mathcal{W}_1$ and $\mathcal{W}_2$ are *separated* if every piece in $\mathcal{W}_1$ is separated from every piece in $\mathcal{W}_2$. An *N-partition* of the cake is a collection of separated portions $\mathcal{W}_1, \ldots, \mathcal{W}_N$ whose union is the entire cake I.

Suppose that the N users $u_1, \ldots, u_N$ wish to share the cake. Each user u_i has a *utility function* $F_i(x)$, which determines how user u_i values the piece $[0, x]$, where $0 \leq x \leq 1$. Each user u_i knows only its own utility function $F_i(x)$, and has no information regarding the utility functions of other users. The functions $F_i(x)$ are monotonically non-decreasing with $F_i(0) = 0$ and $F_i(1) = 1$, for every user u_i. We require that the value of a portion is the sum of the values of the individual pieces in that portion*. Thus, the value of piece $[l, r]$ to user u_i is $F_i([l, r]) = F_i(r) - F_i(l)$, and for any portion $\mathcal{W} = \{P_1, P_2, \ldots P_k\}$, $F_i(\mathcal{W}) = \sum_{i=1}^{k} F_i(P_i)$.

The goal of cake-division is to partition the entire cake I into N separated portions, assigning each user to a portion. Formally, a *cake-division* is an N-partition $\mathcal{W}_1, \ldots, \mathcal{W}_N$ of cake I, with an assignment of portion $\mathcal{W}_i$ to user u_i, for all $1 \leq i \leq N$. Two cake-divisions $\mathcal{W}_1, \ldots, \mathcal{W}_N$ and $\mathcal{W}'_1, \ldots, \mathcal{W}'_N$ are *equivalent* if $\bigcup_{P_i \in \mathcal{W}_j} P_i = \bigcup_{P_i \in \mathcal{W}'_j} P_i$ for all j, i.e., every user gets the same part of cake in both divisions (but perhaps divided into different pieces).

The cake-division is *fair* or *proportional* if $F_i(\mathcal{W}_i) \geq 1/N$, for all $1 \leq i \leq N$, i.e., each user u_i gets what she considers to be at least $1/N$ of the cake according to her own utility function F_i. We obtain the following interesting variations of

* This is a commonly made technical assumption. Practically, there could be situations where a pound of crumbs is not equivalent to a pound of cake.

fair cake-division, if, in addition to fair, we impose further restrictions or *fairness constraints* on the relationship between the assigned portions:

Envy-free: $F_i(\mathcal{W}_i) \geq F_i(\mathcal{W}_j)$ for all i, j; *strong envy-free* if $F_i(\mathcal{W}_i) > F_i(\mathcal{W}_j)$ for all $i \neq j$.

Super envy-free: $F_i(\mathcal{W}_j) \leq 1/n$ for all $i \neq j$; *strong super envy-free* if $F_i(\mathcal{W}_i) < 1/n$ for all $i \neq j$.

These definitions are standard and found in [14].

2.2 Cake-Cutting Algorithms

We now move on to defining a cake-cutting protocol/algorithm. Imagine the existence of some administrator or superuser $\mathcal{S}$ who is responsible for the cake-division. The superuser $\mathcal{S}$ has limited computing power, namely she can perform basic operations such as comparisons, additions and multiplications. We assume that each such basic operation requires one time step.

Superuser $\mathcal{S}$ can ask the users to cut pieces of the cake in order to get information regarding their utility functions. A cut is composed of the following steps: superuser $\mathcal{S}$ specifies to user u_i a piece $[l, r]$ and a ratio R with $0 \leq R \leq 1$; the user then returns the point C in $[l, r]$ such that $F_i([l, C])/F_i([l, r]) = R$. Thus, a cut can be represented by the four-tuple $\langle u_i; [l, r]; R; C \rangle$. We call C the *position* of the cut. It is possible that a cut could yield multiple cut positions, i.e. when some region of the cake evaluates to zero; in such a case we require that the cut position returned is the left-most. In cake-cutting algorithms, the endpoints of the piece to be cut must be either 0, 1, or cut positions that have been produced by earlier cuts. So for example, the first cut has to be of the form $\langle u_{i_1}; [0, 1]; R_1; C_1 \rangle$. The second cut could then be made on $[0, 1]$, $[0, C_1]$ or $[C_1, 1]$. From now every piece will be of this form. We assume that a user can construct a cut in constant time**. A cake-cutting algorithm (implemented by the superuser $\mathcal{S}$) is a sequence of cuts that $\mathcal{S}$ constructs in order to output the desired cake-division.

Definition 1 (Cake-Cutting Algorithm).

Input: *The N utility functions, $F_1(x), \ldots, F_N(x)$ for the users $u_1, \ldots, u_N$.*
Output: *A cake-division satisfying the necessary fairness constraint.*
Computation: *The algorithm is a sequence of steps, $t = 1 \ldots K$. At every step t, the superuser requests a user u_{i_t} to perform a cut on a piece $[l_t, r_t]$ with ratio R_t: $\langle u_{i_t}; [l_t, r_t]; R_t; C_t \rangle$. In determining what cut to make, the superuser may use her limited computing power and the information contained in all previous cuts. A single cut conveys to the super user an amount of information that the superuser would otherwise need to obtain using some comparisons.*

** From the computational point of view, this may be a strong assumption, for example dividing a piece by an irrational ratio is a non-trivial computational task, however it is a standard assumption made in the literature, and so we continue with the tradition.

These comparisons need to also be taken into account in the computational complexity of the algorithm.

We say that this algorithm uses K cuts. K can depend on N and the utility functions F_i. A correct cake-division must take into account the utility functions of all the users, however, the superuser does not know these utility functions. The superuser implicitly infers the necessary information about each user's utility function from the cuts made. The history of all the cuts represents the entire knowledge that $\mathcal{S}$ has regarding the utility functions of the users. By a suitable choice of cuts, $\mathcal{S}$ then outputs a correct cake-division. An algorithm is named according to the fairness constraint the cake-division must satisfy. For example, if the output is fair (envy-free) then the algorithm is called a *fair (envy-free) cake-cutting algorithm.*

A cut as we have defined it is equivalent to a constant number of comparisons. A number of additional requirements can be placed on the model for cake-cutting given above. For example, when a cut is made, a common assumption in the literature is that *every* user evaluates the resulting two pieces for the superuser. Computationally, this assumes that utility function evaluation is a negligible cost operation. For the most part, our lower bounds do not require such additional assumptions. In our discussion we will make clear what further assumptions we make when necessary.

2.3 Particular Algorithms

We briefly present some well known cake-cutting algorithms. More details can be found in [14]. Algorithms A and B are both fair cake-cutting algorithms.

In algorithm A, all the users cut at $1/N$ of the whole cake. The user who cut the smallest piece is given that piece, and the remaining users recursively divide the remainder of the cake fairly. The value of the remainder of the cake to each of the remaining users is at least $1 - 1/N$, and so the resulting division is fair. This algorithm requires $\frac{1}{2}N(N+1) - 1$ cuts.

In algorithm B, for simplicity assume that there are 2^M users (although the algorithm is general). All the users cut the cake at $1/2$. The users who made the smallest $N/2$ cuts recursively divide the left "half" of the cake up to and including the median cut, and the users who cut to the right of the median cut recursively divide the right "half" of the cake. Since all the left users value the left part of the cake at $\geq 1/2$ and all the right users value the right part of the cake at $\geq 1/2$, the algorithm produces a fair division. This algorithm requires $N\lceil \log_2 N \rceil - 2^{\lceil \log_2 N \rceil} + 1$ cuts.

A perfectly legitimate cake-cutting algorithm that does not fit within this framework is the *moving knife fair division algorithm.* The superuser moves a knife continuously from the left end of the cake to the right. The first user (without loss of generality u_1) who is happy with the piece to the left of the current position of the knife yells "cut" and is subsequently given that piece. User u_1 is happy with that piece, and the remaining users were happy to give up that piece. Thus the remaining users must be happy with a fair division of the remaining of the cake. The process is then repeated with the remaining cake and the remaining $N-1$ users. This algorithm makes $N-1$ cuts which cannot be improved upon, since at least $N-1$ cuts need to be made to generate N pieces. However, this algorithm does not fit within the framework we have described, and is an example of a *continuous algorithm*: there is no way to simulate the moving knife with any sequence of discrete cuts. Further, each cut in this algorithm is not equivalent to a constant number of comparisons, for example the first cut conveys the information in $\Omega(N)$ comparisons. Hence, such an algorithm is not of much interest from the computational point of view. The types of algorithms that our framework admits are usually termed *finite* or *discrete* algorithms. More details, including algorithms for envy-free can be found in [14].

3 A Lower Bound for Phased Algorithms

We consider a general class of cake-cutting algorithms, that we call "phased". We find a lower bound on the number of cuts required by phased algorithms that guarantee every user a positive valued portion. *Phased* cake-cutting algorithms have the following properties.

- The steps of the algorithm are divided into *phases*.
- In each phase, every *active* user cuts a piece, the endpoints of which are defined using cuts made during *previous* phases only. In the first phase, each user cuts the whole cake.
- Once a user is assigned a portion, that user becomes inactive for the remainder of the algorithm. (Assigned portions are not considered for the remainder of the algorithm.)

Many cake-cutting algorithms fit into the class of phased algorithms. Typical examples are Algorithms A and B. There also exist algorithms that are not phased, for example Steinhaus' original algorithm.

We say that two algorithms are *equivalent* if they use the same number of cuts for any set of utility functions, and produce equivalent cake-divisions. A piece is *solid* if it does not contain any cut positions – a non-solid piece is the union of two or more separated solid pieces. Our first observation is that any cut by a user on a non-solid piece P giving cut position C can be replaced with a cut by the same user on a solid piece contained in P, yielding the *same* cut position.

Lemma 1. *Suppose that P is the concatenation of separated solid pieces $P_1, \ldots, P_k$, for $k \geq 2$, and that the cut $\langle u_i; P; R; C\rangle$ produces a cut position C. Then, for suitably chosen R' and some solid piece P_m, the cut $\langle u_i; P_m; R'; C'\rangle$ produces the same cut position ($C' = C$). Further, R' and m depend only on R and $F_i(P_1), \ldots, F_i(P_k)$.*

Lemma 1 allows us to restrict our attention to solid piece phased algorithms. The following lemma then gives that an initially solid piece to be cut by some users may be cut in the same spot by each of these users.

Lemma 2. *For any phased algorithm, there are utility functions for which all users who are to cut the same (initially solid) piece will cut at the same position.*

We now give our lower bound for phased algorithms, which applies to any algorithm that guarantees each user a portion of positive value.

Theorem 1 (Lower bound for phased algorithms). *Any phased algorithm that guarantees each of N users a portion of positive value for any set of utility functions, requires $\Omega(N \log N)$ cuts in the worst case.*

The lower bound of $\Omega(N \log N)$ cuts for phased algorithms, demonstrates that even the problem of assigning positive portions to users is non-trivial. This lower bound immediately applies to fair and envy-free algorithms, since these algorithms assign positive portions to users.

4 A Lower Bound for Labeled Algorithms

We present a lower bound on the number of cuts required for a general class of fair algorithms that we refer to as "linearly-labeled & comparison-bounded". The proofs are by reducing sorting to cake-cutting. First, we show that any cake-cutting algorithm can be converted to a *labeled* algorithm which labels every piece in the cake-division. Then, by appropriately choosing utility functions, we use the labels of the pieces to sort a given sequence of integers.

First, we define labeled algorithms and then show how any cake-cutting algorithm can be converted to a labeled one. A *full binary tree* is a binary tree in which every node is either a leaf or the parent of two nodes. A *labeling tree* is a full binary tree in which every left edge has label 0 and every right edge has label 1. Every leaf is labeled with the binary number obtained by concatenating the labels of every edge on the path from the root to that leaf. An example labeling tree is shown in Figure 1. Let v be the deepest common ancestor of two leaves v_1 and v_2. If v_1 belongs to the left subtree of v and v_2 belongs to the right subtree of v, then v_1 is *left* of v_2.

Consider an N-partition $\mathcal{W}_1, \ldots, \mathcal{W}_N$ of the cake. The partition is *labeled* if the following hold:

- For some labeling tree, every (separated) piece P_i in the partition has a distinct label b_i that is a leaf on this tree, and every leaf on this tree labels some piece.
- P_i is left of P_j in the cake if and only if leaf b_i is left of leaf b_j in the labeling tree.

A cake-cutting algorithm is *labeled* if it always produces an N-partition that is labeled. An example of a labeled partition is shown in Figure 1. In general, there are many ways to label a partition, and the algorithm need only output one of those ways. Next, we show that any cake-cutting algorithm can be converted to an equivalent labeled algorithm.

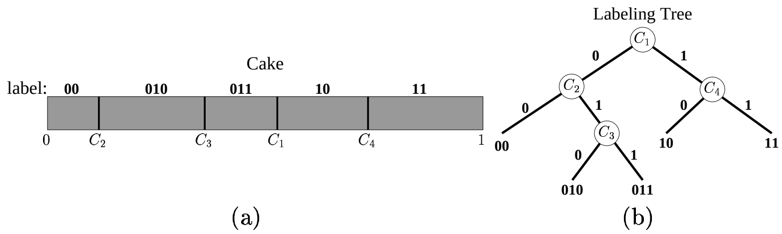

Fig. 1. (a) A labeled partition. (b) Corresponding labeling tree.

Theorem 2. *Every cake-cutting algorithm is equivalent to a labeled cake-cutting algorithm.*

We now show that a labeled cake-cutting algorithm can be used to sort N positive distinct integers $x_1, \ldots, x_N$. To relate sorting to cake-cutting, we first define a "less than" relation for pieces. If P_1 and P_2 are separated, then $P_1 < P_2$ if P_1 is on the left of P_2. Clearly, this "<" relation imposes a total order on any set of separated pieces. Our approach is to show that given N positive distinct integers, we can construct utility functions such that any fair division will allow us to sort the integers *quickly*. Define the utility functions $F_i(x) = \min(1, N^{x_i}x)$, for user u_i. In what follows, F_i will always refer to the utility functions defined above. Let $V_i = 1/N^{x_i}$. Only pieces that overlap $[0, V_i]$ have positive value for user u_i.

Consider any N-partition $\mathcal{W}_1, \ldots, \mathcal{W}_N$, such that each $\mathcal{W}_i$ has a non-zero value for the respective user u_i. Let $R_i \in \mathcal{W}_i$ be the rightmost piece of $\mathcal{W}_i$ that overlaps $[0, V_i]$. The ordering relation on pieces now induces an ordering on portions: $\mathcal{W}_i < \mathcal{W}_j$ if and only if $R_i < R_j$. Next, we show that the order of the portions $\mathcal{W}_i$ is related with the order of the integers x_i.

Lemma 3. *Let $\mathcal{W}_1, \ldots, \mathcal{W}_N$ be a fair cake-division for the utility functions $F_1, \ldots, F_N$. Then, $x_i < x_j$ if and only if $\mathcal{W}_j < \mathcal{W}_i$.*

The ordering relation on portions can be used to sort the N-partition $\mathcal{W}_1, \ldots, \mathcal{W}_N$, i.e., find the sequence of indices $i_1, \ldots, i_N$, such that $\mathcal{W}_{i_1} < \mathcal{W}_{i_2} < \cdots < \mathcal{W}_{i_N}$. An application of Lemma 3 then gives that $x_{i_1} > x_{i_2} > \cdots > x_{i_N}$, thus sorting the partition is equivalent to sorting the integers. We now show that if the partition is labeled, we can use the labels to sort the portions $\mathcal{W}_i$ quickly, which in turn will allow us to sort the integers quickly.

Lemma 4. *Any labeled N-partition $\mathcal{W}_1, \ldots, \mathcal{W}_N$, can be sorted in $O(K)$ time, where K is the total number of pieces in the partition.*

By Lemma 3, sorting the partition $\mathcal{W}_1, \ldots, \mathcal{W}_N$ is equivalent to (reverse) sorting the integers $x_1, \ldots, x_N$. From Lemma 4, we know that if the fair cake-division is labeled, then we can sort the partition in $O(K)$ time, where K is the number of pieces in the partition. Thus, we obtain the following theorem, which reduces sorting to cake-cutting:

Theorem 3 (Reduction of sorting to cake-cutting). *Given a K-piece, labeled, fair cake-division for utility functions $F_1, \ldots, F_N$, we can sort the numbers $x_1, \ldots, x_N$ in $O(K)$ time.*

Theorem 2 showed that every cake-cutting algorithm can be converted to an equivalent labeled cake-cutting algorithm. Of importance is the complexity of this conversion. We say that a cake-cutting algorithm H that outputs a cake-division with K pieces is *linearly-labeled* if it can be converted to a labeled algorithm H' that outputs an equivalent cake division with $O(K)$ pieces using at most $O(K)$ extra time, i.e., if it can be converted to an equally efficient algorithm that outputs essentially the same division. To our knowledge, all the known cake-cutting algorithms are linearly-labeled. In particular, Algorithms A and B can be easily converted to labeled algorithms using at most $O(K)$ additional operations to output an equivalent cake-division. Since sorting is reducible to labeled cake-cutting, labeled cake-cutting cannot be faster than sorting. We have the following result.

Theorem 4 (Lower bound for labeled algorithms). *For any linearly-labeled fair cake-cutting algorithm H, there are utility functions for which $\Omega(N \log N)$ comparisons will be required.*

From the practical point of view one might like to limit the amount of computation the superuser is allowed to use in order to determine what cuts are to be made. Each step in the algorithm involves a cut, and computations necessary for performing the cut. Among these computations might be comparisons, i.e., the superuser might compare cut positions. At step t, let K_t denote the number of comparisons performed. The algorithm is *comparison-bounded* if $\sum_{t=1}^{T} K_t \leq \alpha T$ for a constant α and all T. Essentially, the number of comparisons is linear in the number of cuts. The labeled algorithms A are B are easily shown to be comparison-bounded. We now give our lower bound on the number of cuts required for linearly-labeled comparison-bounded algorithms.

Theorem 5 (Lower bound for comparison-bounded algorithms). *For any linearly-labeled comparison-bounded fair algorithm H, utility functions exist for which $\Omega(N \log N)$ cuts will be made.*

5 Lower Bounds for Envy-Free Algorithms

We give lower bounds on the number of cuts required for strong and super envy-free division, when such divisions exist. We show that there exist utility functions that admit acceptable divisions for which $\Omega(N^2)$ cuts are needed.

Theorem 6 (Lower bound for strong envy-free division). *There exist utility functions for which a strong envy-free division requires $\Omega(0.086N^2)$ cuts.*

Theorem 7 (Lower bound for super envy-free division). *There exist utility functions for which a super envy-free division requires $\Omega(0.25N^2)$ cuts.*

6 Concluding Remarks

The most general results are that any cake-cutting algorithm can be converted to a solid piece algorithm, and then to a labeled algorithm. We then showed that any labeled fair cake-cutting algorithm can be used to sort, therefore any fair cake-cutting algorithm can be used to sort. This provided the connection between sorting and cake-cutting. We also provided an independent strong result for phased algorithms, namely that $\Omega(N \log N)$ cuts are needed to guarantee each user a positive valued portion, and we also obtained $\Omega(N^2)$ bounds for two types of envy-free division. Important open questions remain and we refer the reader to the technical report [10] and the literature for more details.

References

1. J. Barbanel. Game-theoretic algorithms for fair and strongly fair cake division with entitlements. *Colloquium Math.*, 69:59–53, 1995.
2. J. Barbanel. Super envy-free cake division and independence of measures. *J. Math. Anal. Appl.*, 197:54–60, 1996.
3. Steven J. Brams and Allan D Taylor. An envy-free cake division protocol. *Am. Math. Monthly*, 102:9–18, 1995.
4. Steven J. Brams and Allan D. Taylor. *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press, New York, NY, 1996.
5. Stephen Demko and Theodore P. Hill. Equitable distribution of indivisible objects. *Mathematical Social Sciences*, 16(2):145–58, October 1988.
6. L. E. Dubins and E. H. Spanier. How to cut a cake fairly. *Am. Math. Monthly*, 68:1–17, 1961.
7. Jacob Glazer and Ching-to Albert Ma. Efficient allocation of a 'prize' – King Solomon's dilemma. *Games and Economic Behavior*, 1(3):223–233, 1989.
8. C-J Haake, M. G. Raith, and F. E. Su. Bidding for envy-freeness: A procedural approach to n-player fair-division problems. *Social Choice and Welfare*, To appear.
9. Jerzy Legut and Wilczýnski. Optimal partitioning of a measuarble space. *Proceedings of the American Mathematical Society*, 104(1):262–264, September 1988.
10. Malik Magdon-Ismail, Costas Busch, and Mukkai Krishnamoorthy. Cake-cutting is not a piece of cake. Technical Report 02-12, Rensselaer Polytechnic Institute, Troy, NY 12180, USA, 2002.
11. Elisa Peterson and F. E. Su. Four-person envy-free chore division. *Mathematics Magazine*, April 2002.
12. K. Rebman. How to get (at least) a fair share of the cake. *in Mathematical Plums (Edited by R. Honsberger), The Mathematical Association of America*, pages 22–37, 1979.
13. Jack Robertson and William Webb. Approximating fair division with a limited number of cuts. *J. Comp. Theory*, 72(2):340–344, 1995.
14. Jack Robertson and William Webb. *Cake-Cutting Algorithms: Be Fair If You Can.* A. K. Peters, Nattick, MA, 1998.
15. H. Steinhaus. The problem of fair division. *Econometrica*, 16:101–104, 1948.
16. F. E. Su. Rental harmony: Sperner's lemma in fair division. *American Mathematical Monthly*, 106:930–942, 1999.
17. Gerhard J. Woeginger. An approximation scheme for cake division with a linear number of cuts. In *European Symposium on Algorithms (ESA)*, pages 896–901, 2002.