# Cake Cutting Really Is Not a Piece of Cake



JEFF EDMONDS, York University
KIRK PRUHS, University of Pittsburgh

We consider the well-known cake cutting problem in which a protocol wants to divide a cake among $n \geq 2$ players in such a way that each player believes that they got a fair share. The standard Robertson-Webb model allows the protocol to make two types of queries, Evaluation and Cut, to the players. A deterministic divide-and-conquer protocol with complexity $O(n \log n)$ is known. We provide the first a $\Omega(n \log n)$ lower bound on the complexity of any deterministic protocol in the standard model. This improves previous lower bounds, in that the protocol is allowed to assign to a player a piece that is a union of intervals and only guarantee approximate fairness. We accomplish this by lower bounding the complexity to find, for a single player, a piece of cake that is both rich in value, and thin in width. We then introduce a version of cake cutting in which the players are able to cut with only finite precision. In this case, we can extend the $\Omega(n \log n)$ lower bound to include randomized protocols.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms: Algorithms, Economics

Additional Key Words and Phrases: Cake cutting, fair division

**ACM Reference Format:**
Edmonds, J. and Pruhs, K. 2011. Cake cutting really is not a piece of cake. ACM Trans. Algor. 7, 4, Article 51 (September 2011), 12 pages.
DOI = 10.1145/2000807.2000819 http://doi.acm.org/10.1145/2000807.2000819

## 1. INTRODUCTION

Our setting is a collection of self-interested entities who desire to partition a disparate collection of items of value. Imagine heirs of an estate wanting to divide the possessions of the newly departed. Or imagine the creditors of a bankrupt company, such as Enron, wanting to split up the company's remaining assets. The entities may well value the items differently. For example, one can imagine different heirs of an estate not necessarily agreeing on the relative value a baseball signed by Pete Rose, a worn leather lazy boy recliner, a mint condition classic Farrah Fawcett poster, etc. The goal is devise a protocol to split up the items fairly, that is, so every entity believes that he/she gets a fair share based on how he/she values the objects. Achieving this goal is potentially complicated by the fact that the entities may well be greedy, deceitful, treacherous, etc. They may not be honest about how they value the objects, they may

J. Edmonds was supported in part by NSERC Canada. K. Pruhs was supported in part by an IBM faculty award and by NSF grants CNS-0325353, CCF-0514058, IIS-0534531, and CCF 0830558.
Authors' addresses: J. Edmonds, Department of Computer Science and Engineering, York University, CSEB 3044, 4700 Keele Street, Toronto, Ont., Canada M3J 1P3; email: jeff@cse.yorku.ca; K. Pruhs, Department of Computer Science, University of Pittsburgh, 210 South Bouquet Street, Sennott Square Building, Room 6415, Pittsburgh, PA 15260, email; kirk@cs.pitt.edu.

DOI 10.1145/2000807.2000819 http://doi.acm.org/10.1145/2000807.2000819

collude together to cheat another entity, etc. So we seek a protocol that guarantees a fair share to everyone that is honest. If someone tries to cheat or lie, then they cannot blame the protocol if they don't end up with a fair share.

In the literature, this problem falls under the rubric of cake cutting [Brams and Taylor 1996; Robertson and Webb 1998]. (This is motivated by the well-known phenomenon that some people value the frosting more than others.) The cake cutting problem arose from the 1940s school of Polish mathematicians. Since then the problem has blossomed and been widely popularized [Robertson and Webb 1998]. Most people find cake cutting problems psychologically and socially interesting, and some quick Googling reveals that cake cutting algorithms, and their analyses, are commonly covered in algorithms and discrete mathematics courses.

Cake cutting in formalized in the following manner. The objects of value are ordered in some arbitrary way, and then abstracted away into subintervals of the interval $[0, 1]$, which is the cake. Each entity/player has a value function $V$ that specifies how much that player values a particular subinterval, or more precisely, the objects in the subinterval. This is a reasonable model if the value of each item is small relative to the total value of the items. The protocol can query players about their value functions, which are initially unknown to the protocol. In the standard Robertson-Webb model [Robertson and Webb 1998; Sgall and Woeginger 2003], the two types of queries are Evaluation and Cut. In an Evaluation query, a player is asked how much he values a subinterval. In a Cut query, the player is asked to identify an interval, with a fixed left endpoint, of a particular value.

### 1.1. Previous Results

Sgall and Woeginger [2003] provide a nice brief overview of results in this area. Books by Robertson and Webb [1998] and Brams and Taylor [1996] provide more extensive overviews.

Let us first consider upper bound results. A deterministic protocol that uses $\Theta(n^2)$ cuts was described by Steinhaus [1948]. Evan and Paz [1984] gave a deterministic divide and conquer protocol that has complexity $\Theta(n \log n)$. Further, they gave a randomized protocol that uses $\Theta(n)$ cuts and $\Theta(n \log n)$ evaluations.

Approximately fair protocols were introduced by Robertson and Webb [1995]. We say that a protocol is *c-fair* if it guarantees each honest player a piece of cake that he believes has value at least $1/cn$. There is a deterministic protocol that achieves $O(1)$-fairness with $\Theta(n)$ cuts and $\Theta(n^2)$ evaluations [Robertson and Webb 1995; Krumke et al. 2002; Woeginger 2002].

Traditionally, much of the research has focused on minimizing the number of cuts, without too much regard for the number of evaluations. In the settings that we are interested in, e.g. heirs splitting an inheritance, there is no good reason to assume that evaluation queries are especially easier or cheaper than cut queries. It is not clear why the initial focus was on minimizing cuts. One possibility is concern that too many cuts would lead to crumbling of a literal cake. In any case, we will view evaluation and cut queries as equally expensive, and define the complexity of a protocol to be the number of queries used.

Thus, one can summarize the known upper-bound results as follows: There is a deterministic protocol with complexity $O(n \log n)$ that guarantees exact fairness. No protocol that uses a linear number of queries is known, even if randomization is allowed, and even if the protocol need only guarantee $O(1)$-fairness.

So a natural avenue for investigation is to attempt to prove an $\Omega(n \log n)$ lower bound on the complexity of any cake cutting protocol. The most obvious way to prove such a lower bound is to try to reduce sorting (or more precisely, learning an unknown permutation) to cake cutting. A first step in this direction was taken by

Magdon-Ismail et al. [2003], who were able to show that any protocol must make $\Omega(n \log n)$ comparisons to compute the assignment. So this result did not really lower bound the number of queries. A second step in this direction was taken by Sgall and Woeginger [2003] who give a more complicated reduction from sorting to show an $\Omega(n \log n)$ lower bound on the complexity of any deterministic protocol that is required to assign to each player a piece that is a single subinterval of the cake. On the positive side, all known protocols have this property. On the other hand, there is no natural reason to impose this restriction in the settings that we are interested in. That is, it is perfectly reasonable to assign to an inheritor a collection of items that are not consecutive in the initial arbitrary ordering of the items. The lower bound of Sgall and Woeginger [2003] can be seen to hold against randomized protocols. However, note that neither of these lower bounds [Magdon-Ismail et al. 2003; Sgall and Woeginger 2003] hold if the protocol is only required to achieve approximate fairness.

### 1.2. Our Results

In Section 2, we give a lower bound of $\Omega(n \log n)$ on the complexity of any deterministic protocol for cake cutting, which is the first $\omega(n)$ lower bound in the general Robertson-Webb model. Recall that the complexity of a protocol is the number of evaluation and cut queries used. Our lower bound improves on the results in Sgall and Woeginger [2003] in two ways: (1) it applies to protocols that may assign to a player a piece that is a union of intervals, and (2) it applies to protocols that only guarantee $n^{1-\delta}$ approximate fairness, that is, players can be allocated pieces with value as low as $1/cn = 1/n^{2-\delta}$.

We believe that the main reasons why earlier lower bounds were not stronger is that they essentially attempted to reduce from sorting, which does seem to capture the difficulty of cake cutting in the general model. Instead, we observe that not only are the players required to find a piece that is rich in value, but if their pieces are not to overlap then most players need a piece that is thin in width. We obtain our $\Omega(n \log n)$ lower bound by showing a lower bound of $\Omega(\log n)$ on the complexity of a single player finding a piece that is both thin and rich, where thin means that the width at most $2/n$, and rich means that the value at least $1/cn$. It is easy to see how to find a piece that is thin and rich in $O(1)$ time using a randomized algorithm. With probability at least $1/2$, a random interval of width $1/n$ is thin and rich. Thus, our deterministic lower bound does not extend to randomized algorithms.

To our knowledge, all the literature to date has assumed that players can answer cut and evaluation queries with exact precision. This is probably not so realistic in some settings, for example, it is probably too much to ask an inheritor to value an arbitrary subcollection of items to within a penny. For this reason, we introduce what we call approximate cut queries to which a player need only return an interval of cake of value within a $1+\epsilon$ factor of the requested value. To our knowledge, no one to date has considered approximate queries.

In Section 3, we prove that if $\epsilon$ is a constant, then there is an $\Omega(n \log n)$ lower bound on the complexity of any randomized protocol for cake cutting with approximate cuts (with relative error $1 + \epsilon$) and exact evaluation queries, even if only $n^{1-\delta}$-fairness is required. The fact that the protocol is allowed exact evaluations, but only approximate cuts, demonstrates the asymmetric power of these two operations. This lower bound is oblivious in that our adversary doesn't change the lower bound instance in response to random events internal to the protocol.

We believe that the main contribution of this article, beyond the explicit lower bounds, is the identification of the importance of the problem of finding thin rich pieces. We also believe that the concept of approximate queries is interesting, and worthy of further investigation.

### 1.3. Formal Problem Statement

The cake consists of the interval $[0, 1]$. Each player $p$, $1 \leq p \leq n$, has value function $V_p(x_1, x_2)$ which specifies a value in the range $[0, 1]$ that a player assigns to the subinterval $[x_1, x_2]$. Player values are scaled so that they each have value 1 for the whole cake, that is, $V_p(0, 1) = 1$. The value function should be additive, that is, $\forall x_1 \leq x_2 \leq x_3 \in [0, 1]$, $V_p(x_1, x_2) + V_p(x_2, x_3) = V_p(x_1, x_3)$. Further, the value function must give value 0 to every point, that is, $V_p(x, x) = 0$. In this article, a piece of cake is a collection of subintervals, not necessarily a single subinterval. Further, the ends of each subinterval in a piece must have been at one of the ends of a cut. The value of a piece of cake is then just the sum of the values of the subintervals of the piece. The value functions are initially unknown to the protocol.

The protocol's goal is to assign to each player $p$ a piece $C_p$ of the cake. The pieces must be disjoint, that is, $C_p$ and $C_q$ must be disjoint for all players $p \neq q$. Further, the protocol should be *c-fair* to each player $p$, that is, it must be the case that the value of $C_p$ according to $V_p$ is at least $c/n$. Thus one gets different variations to the problem depending on the value of $c$.

In order to achieve its goal in the Robertson and Webb model, the protocol may repeatedly ask any player one of two types of queries.

—$AEval_p(\epsilon, x_1, x_2)$. This $1 + \epsilon$ approximate evaluation query to player $p$ returns an $(1+\epsilon)$-approximate value of the interval $[x_1, x_2]$ of the cake for player $p$. That is, $\frac{1}{1+\epsilon} V_p(x_1, x_2) \leq AEval_p(\epsilon, x_1, x_2) \leq (1 + \epsilon)V_p(x_1, x_2)$. An exact evaluation query, $Eval_p(x_1, x_2)$, is equivalent to $AEval_p(0, x_1, x_2)$.
—$ACut_p(\epsilon, x_1, \alpha)$. The $1 + \epsilon$ approximate cut query returns an $x_2 \geq x_1$ such that the interval of cake $[x_1, x_2]$ has value approximately $\alpha$ according to player $p$'s value function $V_p$. More precisely, $x_2$ satisfies $1/(1+\epsilon)V_p(x_1, x_2) \leq \alpha \leq (1+\epsilon)V_p(x_1, x_2)$. An exact cut query, $Cut_p(x_1, \alpha)$, is equivalent to $ACut_p(0, x_1, \alpha)$.

The protocol may be adaptive in the sense that the protocol need only decide on the $i$th query after it has seen the outcome of the first $i - 1$ queries and when randomized on coin flips. The complexity of a protocol is the worst-case, over all possible valuation functions, of the expected number of queries needed to accomplish its goal. For cake cutting, Las Vegas and Monte Carlo algorithms are of equal power; Since the complexity of verifying the correctness of an assignment has linear complexity, a Monte Carlo algorithm can be converted into a Las Vegas algorithm.

As Sgall and Woeginger [2003] point out, cut and evaluation queries can efficiently simulate all other types of queries used in protocols in the literature, for example, cutting the cake into two parts with a specified ratio of value. There are many technical issues that must be considered when formally defining the "right" model. A nice discussion of these issues can be found in Sgall and Woeginger [2003]. For example, in the standard model, after a cut, each piece is re-indexed to $[0, 1]$. As we are proving lower bounds, it is more convenient to continue to index with respect to the entire cake. Several issues that are relevant when proving upper bounds—for example, further niceness properties on the value functions, and robustness against cheating—are not particularly relevant to us here. Our value functions satisfy every niceness property considered in the literature. To prove our lower bound, it suffices to consider only the case when all players are honest. Our lower bounds are robust against reasonable minor modifications to the model.

## 2. THE DETERMINISTIC LOWER BOUND

This section is devoted to showing an $\Omega(n \log n)$ lower bound on the complexity of cake cutting for deterministic algorithms. We first introduce a new game, that we call the thin-rich game, which takes place in the same setting as the cake cutting game. We then introduce what we call value trees, which represent certain types of cake value functions that will be useful for us. We then give an adversarial strategy for the thin-rich game on value trees. This adversarial strategy gives a bound of $\Omega(\log n - \log c)$ on the complexity of the thin-rich rich game. We then show that any protocol for cake cutting has to essentially solve $\Theta(n)$ different thin-rich games, giving us a lower bound of $\Omega(n(\log n - \log c))$ on the complexity of cake cutting.

*Thin-Rich Game.* This game involves a single player. We say that a piece of cake is *thin* if it has width at most $2/n$. We say a piece is *rich* if it has value at least $1/cn$ for this player. The goal for the protocol is to identify a thin rich piece of cake.

We now define *value trees* and explain how a value function is derived from a value tree. Assume $n \geq 6$ is twice an integer power of 3. The tree is a balanced 3-ary rooted tree with $n/2$ leaves, depth $L = \log_3 n/2$, and a value $V(u)$ for each node. For each internal node $u$, its left, middle, and right children are denoted $l(u)$, $m(u)$, and $r(u)$. Two of these three edges are labeled 1/4 and are called *light edges*, and the remaining edge is labeled 1/2 and is called a *heavy edge*. The value $V(u)$ of node is the product of the edge labels along the path from the root to $u$. Note that $u$'s value is the sum of its children's values, that is, $V(u) = (1/4)V(u) + (1/4)V(u) + (1/2)V(u) = V(l(u)) + V(m(u)) + V(r(u))$. Let $d(u)$ denote the number of edges in the path from the root to the node $u$. Let $q(u)$ denote the number of these that are heavy edges. It follows that $V(u) = (1/2)^{q(u)}(1/4)^{d(u)-q(u)}$.

The cake is partitioned into $n/2$ *thin* intervals, namely for $i \in [1, n/2]$, the $i$th interval of width $\frac{2}{n}$ is $[2(i-1)/n, 2i/n]$. These $n/2$ intervals are associated with the leaves of the value tree. We associate with each internal node $u$, the interval of cake that is the union of the leaves of the subtree rooted at $u$. We say that a point is in a node in the tree if it is contained in the interval association with that node. The width of this interval is $W(u) = 3^{-d(u)}$, and its value is given by $V(u)$. If $u$ is a leaf, then this value is spread evenly over this interval.

Some intuition may be useful. The canonical thin-rich piece, which is the goal of a protocol to find, consists a leaf $u$ with density $D(u) = V(u)/W(u) = 1/cn/2/n = 2/c$. Towards this goal, the protocol must find nodes in the tree that are both low in the tree and dense. In order for a node to have high density, the path from the root to it must have lots of heavy edges, namely

$$D(u) = \frac{V(u)}{W(u)} = \frac{(1/2)^{q(u)}(1/4)^{d(u)-q(u)}}{(1/3)^{d(u)}} \geq \frac{2}{c}$$

Or equivalently,

$$q(u) \geq \log_2\left(\frac{4}{3}\right) d(u) - \log_2 c > \frac{4}{10}\, d(u) - \log_2 c$$

The obvious $O(\log n)$ time protocol, starts at the root, which has density $D(u) = 1$, and follows the unique path consisting of only heavy edges down the tree. If a deterministic protocol attempts to circumvent this process by leaping to a lower node, then the adversary can simply fix the edges in the path to this node to be light. If a randomized protocol selects a random node, then each edge is heavy with probability 1/3 giving $q(u) = (1/3)d(u)$, which is much less than the $q(u) = (4/10)d(u) - \log_2 c$ needed for it to be rich.

We say that a protocol for the thin-rich game is *normal* if, when the value function is derived from a value tree, the protocol always returns a leaf of the value tree. We now show that, without loss of generality, we may restrict our attention to normal protocols.

LEMMA 2.1. *If there is a deterministic protocol A for the thin-rich game with complexity $T(n)$, then for value functions derived from value trees, there is a normal deterministic protocol B with complexity $O(T(n))$.*

PROOF. Consider an arbitrary protocol $A$. Let $\mathcal{I}$ denote the collection of intervals returned by $A$. Because overall $\mathcal{I}$ has density at least $2/c$, at least one $I$ of these intervals in $\mathcal{I}$ does as well. Since the cardinality of $\mathcal{I}$ is at most $T(n)$, one such interval $I$ can be found with $O(T(n))$ queries. Because this interval has width at most $2/n$, it overlaps with at most two leaves of the value tree. Because each leaf has uniform value along its width, at least one of these two leaves must have density at least $2/c$. The protocol $B$ can query the value of each of these two leaves, and return the one of higher value. □

As a protocol for the thin-rich game asks queries, it gains information about the value tree. In order to bound the information learned, the lower bound adversary reveals the labels of enough edges of the value tree to provide the protocol with at least as much information as the thin-rich protocol gets from the query. Let $P = u_0, \ldots, u_k$ be a path from the root $u_0$ of the value tree to a node $u_k$. The node $u_k$ is said to be *revealed* if all the labels on all the edges leading from a node $u_i$, $0 \leq i \leq k-1$, to a child of $u_i$, are revealed. Lemma 2.2 quantifies what can be learned from revealed vertices.

LEMMA 2.2

—*For any revealed node u, the value $V(u)$ of the interval of cake under it can be computed.*
—*Let u be a revealed node, let x be the leftmost point in u, and y the rightmost point in u. Then $V(0, x)$ and $V(0, y)$ can be computed.*
—*Let x be a point in a revealed leaf u, and let $y \geq x$ be a point in a revealed leaf v. Then $V(0, x)$ and $V(x, y)$ can be computed.*
—*Let u be a revealed leaf, let $x_1$ is a point in u, and let $\alpha$ be a cake value. From this information, the least common ancestor of u and the node v that contains the point $x_2$ satisfying $V(x_1, x_2) = \alpha$ can be computed.*

PROOF. We consider the items one by one. For the first item, $V(u)$ is just the product of the edge labels leading to $u$. Consider the second item. Let $u_0, \ldots, u = u_k$ be the path from the root to $u$. $V(0, x)$ is then just the sum of the values of the siblings to the left of a $u_i$, $1 \leq i \leq k$, which may be computed by the previous item. $V(0, y)$ is then $V(0, x) + V(u)$. Consider the third item. Let $x'$ be the left most point in the leaf $u$. Because the value of the cake is uniform on leaves, $V(x', x) = (x - x')/(2/n) \cdot V(u)$. Then $V(0, x) = V(0, x') + V(x', x)$ and $V(x, y) = V(0, y) - V(0, x)$. Consider the fourth item. Let $u_0, \ldots, u_L = u$ be the path from the root to the leaf $u$ containing $x_1$. Proceed up the tree from $u$, computing $V(x_1, y_i)$ where $y_i$ is the right most point under $u_i$. Here $V(x_1, y_L) = (y_L - x_1)/(2/n) \cdot V(u)$ and $V(x_1, y_i)$ is $V(x_1, y_{i+1})$ plus the sum the values of the children of $u_i$ that are to the right of $u_{i+1}$. When the sum exceeds $\alpha$, then $u_i$ is the least common ancestor. □

We are now ready to give the lower bound for the thin-rich game.

LEMMA 2.3. *The deterministic complexity of the thin-rich game is $\Omega(\log n - \log c)$.*

PROOF. We give an adversarial strategy that lower bounds the complexity of an arbitrary normal protocol $A$ on a value tree distribution. We maintain a number of invariants. First, for each node, either none or all three of its outgoing edges have been revealed. Second, at each point in time the set of revealed nodes forms of a connected component of the value tree that contains the root. Third, after $k$ queries, any root to leaf path contains at most $2k$ edges revealed to be heavy. Initially, these invariants are trivially true.

Suppose that on the $k$th query, the protocol $A$ makes the query $Eval(x_1, x_2)$. Let $u_0, \ldots u_L$ be the path from the root to the leaf containing $x_1$. Let $u_i$ be the highest unrevealed node in this path. The edges in $u_i, \ldots u_L$ are revealed to be light. For each of these nodes, when its outgoing edge is revealed to be light, one of its other two outgoing edges is revealed to be light and the other heavy. Note that this automatically reveals not only all the nodes in this path, but also reveals all the children of the nodes in this path. This same process is then repeated for $x_2$. By Lemma 2.2, $A$ will then have enough information to compute the answer to this *Eval* query. It is easy to verify that the invariants are maintained.

Now suppose that on the $k$th query, the protocol $A$ makes the query $Cut(x_1, \alpha)$. As done for an *Eval* query, the "forked" path from the root to the leaf containing $x_1$ is revealed. As given by Lemma 2.2, let $u_i$ be the least common ancestor of the leaf containing $x_1$ and the leaf containing the unknown point $x_2$ satisfying $V(x_1, x_2) = \alpha$. We will now describe how to recursively determine and reveal the path $U$ from $u_i$ to $x_2$ in such a way that all these edges on this path are light. Starting from $u_i$, follow revealed edges down in the tree toward the leaf that contains $x_2$ until one reaches an unrevealed node $v$. Let $\beta$ be the value of cake that $A$ seeks from the subtree rooted at $v$. This is well defined by Lemma 2.2. If $\beta/V(v) \leq 1/2$, then the three edges leading to children of $v$ are labeled [1/4, 1/4, 1/2]; otherwise, they are labeled [1/2, 1/4, 1/4]. Note that either way, the next edge in $U$ will be a light edge. Now we redefine $v$ to be the first unrevealed vertex on the path from the root to the leaf containing $x_2$, and repeat this process. This process ends when $v$ becomes the leaf containing $x_2$. By Lemma 2.2, $A$ now has enough information to determine the value of $x_2$. It is easy to verify that all invariants are maintained.

Suppose that the protocol terminates after $((4/10)L - \log_2 c)/2$ queries claiming that a leaf node $u$ is rich. The second invariant states that at most $(4/10)L - \log_2 c$ edges on the path from the root to $u$ have been revealed to be rich. By making the rest of the edges in this path light, we can make $u$ not rich. This then contradicts the correctness of $A$. $\square$

We now give the lower bound for cake cutting.

THEOREM 2.4. *The complexity of any deterministic protocol for cake cutting is* $\Omega(n(\log n - \log c))$*, even with exact queries and only* $c$*-approximate fairness is required. Note that this bound is* $\Omega(n \log n)$ *even when* $c = n^{1-\delta}$.

PROOF. We give an adversarial strategy against an arbitrary deterministic protocol $A$. Let $Q$ be an arbitrary query made by $A$. Assume that query $Q$ is addressed to player $p$. Let $Q_1, \ldots, Q_k$ be the previous queries made by $A$ to player $p$. Then, the adversarial strategy answers $Q$ in the same way that the adversarial strategy for the thin-rich game in Lemma 2.3 would have answered query $Q$ given prior queries $Q_1, \ldots, Q_k$. Now assume to reach a contradiction that $A$ stops before it has asked at least $n/2$ of the players $\Omega(\log n - \log c)$ queries (the lower bound for the thin-rich game from Lemma 2.3). If at least half the pieces that $A$ assigns the players are not thin, then this is not a feasible solution since some pieces must intersect. Otherwise, consider a player that was assigned a thin piece, but whom was queried less than $\Omega(\log n - \log c)$ times (the lower bound for the thin-rich game from Lemma 2.3). By the correctness of the adversarial strategy from Lemma 2.3), there is value function consistent with all of the query answers for this player, but for which this piece is not rich. This contracts the correctness of $A$. $\square$

### 3. THE RANDOMIZED LOWER BOUND

This section is devoted to proving an $\Omega(n \log(n/c))$ lower bound for randomized cake cutting algorithms that only receive approximate answers to their queries. We first use Yao's technique to give a $\Omega(\log(n/c))$ lower bound for the thin-rich game. Yao's technique states that it is sufficient to exhibit an input distribution on which the average-case complexity of every deterministic protocol is at least the lower bound that one wants to prove. Our input distribution, chooses independently for each player a random value tree from which to derive his value function. This is done by choosing independently for each node in the tree, one of its outgoing edges to be heavy. We again then reduce cake cutting to the thin-rich game.

As in the deterministic lower bound, we may restrict our attention to normal thin-rich protocols, that is, those that return a leaf in the value tree. We now introduce a game, the path and triangle game, that we show captures the complexity of finding path in the value tree that is sufficiently rich in heavy edges to give a rich leaf.

*Definition of the Path and Triangle Game.* The protocol is given a value tree, except that it does not know the value of the labels. The protocol makes a sequence of queries, where each query is either a path query or a triangle query. Both types of queries specify a node $u$ in the tree. In response to a path query, the labels on all of the edges incident to a node on the path from the root to $u$ are revealed to the protocol. In response to a triangle query, the labels on all the edges, on all the paths leading from $u$ to descendants of $u$, up to depth $\gamma = 2 + \log_2(\frac{1}{\epsilon})$, in the subtree rooted at $u$, are revealed to the protocol. The protocol's goal is to find a rich path, that is, one with at least $(4/10)L - \log_2 c$ heavy edges. The complexity of a particular protocol is the number of path and triangle queries needed to accomplish this goal.

We now show how to reduce the thin-rich game to the path and triangle game.

LEMMA 3.1. *If the complexity of the path and triangle game is lower bounded by $T(n)$ for a random value tree, then the complexity of thin-rich game is $\Omega(T(n))$ when the value function is derived from a random value tree.*

PROOF. We will prove the contrapositive, that is, a thin-rich protocol $A$ with complexity $T(n)$ implies the existence of a protocol $B$ for the the path and triangle game with complexity $O(T(n))$. We construct $B$ by simulating $A$.

Suppose that protocol $A$ makes the query $AEval(\epsilon, x_1, x_2)$. We will graciously provide protocol $A$ with an answer to the exact query $Eval(x_1, x_2)$. Protocol $B$ then makes two path queries: one query to the leaf containing $x_1$ and one query to the leaf containing $x_2$. The value $V(x_1, x_2)$ can then be computed by Lemma 2.2, and is then returned to $A$ as the result to the $Eval$ query.

Suppose that protocol $A$ makes the query $ACut(\epsilon, x_1, \alpha)$. Let $x_2$ denote the point that $A$ seeks, that is the point such that $V(x_1, x_2) = \alpha$. Note that at this point in time, neither $A$ nor $B$ may know the exact value of $x_2$, but nevertheless we wish to reason about $x_2$. Protocol $B$ then makes at most two path queries and at most one triangle query. After these queries, protocol $B$ will have enough information to provide protocol $A$ with a point $y$ such that $V(x_1, y)$ is sufficiently close to $\alpha$. We now define these three queries. Figure 1 may be useful in understanding the queries.

*The First Path Query.* Let $u_0, u_1, \ldots, u_L$ be the sequence of nodes along the path from the root to the leaf containing $x_1$ in the tree. The first path query is to the node $u_L$. If both $x_1$ and $x_2$ are in $u_L$ then $B$ may return $x_2$. If $x_2$ is not in the leaf $u_L$, then $B$ makes a second path query.

*The Second Path Query.* Using Lemma 2.2, Protocol $B$ computes the least common ancestor $u_r$ of the leaf containing $x_1$ and the leaf containing $x_2$. All of the children of

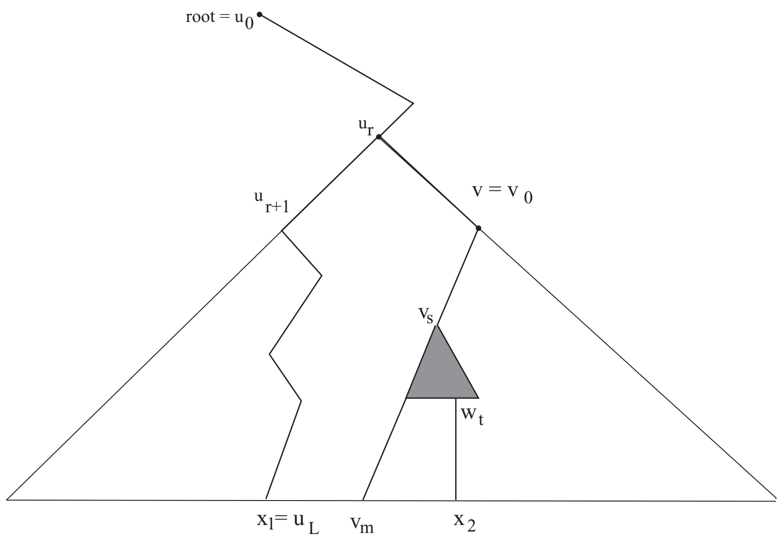


Fig. 1. The two path queries and one triangle query associated with an approximate cut.

$u_r$ must be revealed since $u_{r+1}$ is on the revealed path from the root to $x_1$. Let $v$ be the child of $u_r$ containing $x_2$. The second path query is to the leftmost leaf $v_m$ in the subtree rooted at $v$. Let $v = v_0, v_1, \ldots, v_m$ be the nodes along the path from $v$ to $v_m$.

*The Triangle Query.* Again, using Lemma 2.2, Protocol $B$ computes the least common ancestor $v_s$ of $v_m$ and the leaf containing $x_2$. The triangle query is to the node $v_s$.

*Computing the Result.* If the height of $v_s \leq \gamma$, then the leaf containing $x_2$ is known to $B$. The value of $x_2$ can then can be computed by Lemma 2.2 and is returned to protocol $A$. Otherwise, let $w_1, w_2, \ldots, w_{2^\gamma}$ be the descendants of $v_s$ of depth $\gamma$ in the subtree rooted at $v_s$. Let $w_t$ be the node such that $x_2$ is in the subtree rooted at $w_t$. The point $y$ returned by protocol $B$ will any point in the interval $w_t$.

We now argue the correctness of the result returned by protocol $B$. Because both $x_2$ and $y$ are under $w_t$, the error $V(x_1, y) - \alpha$ will be at most $V(w_t)$. We have $V(w_t) \leq (1/2)^\gamma V(v_s)$, since $w_t$ is $\gamma$ edges below $v_s$ and every edge has a label of at most $1/2$. We have $V(v_s) \leq 4V(l(v_2s)$, because the edge to the left child $l(v_s)$ of $v_s$ has a label of at least $1/4$. We have $4(1/2)^\gamma \leq \epsilon$, by the definition of $\gamma = 2 + \log_2(1/\epsilon)$. We have $V(l(v_2s) \leq \alpha$, since the interval under $l(v_s)$ is totally contained in the interval of value $\alpha$ to the right of $x_1$. Combining these gives that $V(w_t) \leq \epsilon\alpha$. This completes simulation of the $ACut(\epsilon, x_1, \alpha)$ query.

In the end, protocol $A$ finds a rich leaf, which provides protocol $B$ with a rich path. □

We now wish to prove that with probability $\Omega(1)$ the complexity of every randomized protocol for the path and triangle game is $\Omega(\log \frac{n}{\epsilon} / \log \frac{1}{\epsilon})$ if the input is a random value tree. Let *Det* be the set of nodes $u$ which have been revealed, that is, labels on path to $u$ are known. We define a potential function $F(u)$ on a node $u$ by $((11/4)q(u) - d(u))$. Note that for every node, it is the case that in a random value tree, $q(u) = (1/3)d(u)$ and $F(u) = (11/4)(1/3)d(u) - d(u) = -(1/12)d(u)$, but for a rich path $q(u) \approx (4/10)d(u)$ and $F(u) \approx (11/4)(4/10)L - L = (1/10)L$. We define a potential function $F$ for the state of the game by $F = \max_{u \in Det} F(u)$. Initially, *Det* consists of only the root. As $d(root) = 0$, and $q(root) = 0$, it is the case that initially $F(root) = 0$. Also note that $F(u)$ is nonnegative for every node. We now bound the expected change in the value of the potential function as the result of a single query.

LEMMA 3.2. *There exists a constant $\beta$, such that the expected change in $F$ as the result of one query is at most $2\gamma + \beta$.*

PROOF. First, consider the path operation. The player specifies one leaf $x$ and learns the labels on the path $U$ from the root to $x$ (plus the labels on the other edges that lead from a node $y$ on $U$ to a child of $y$). Let $u$ be the last node in $U$ that was in *Det* before the path query. Let $v$ be the node for which $F$ is maximized after the path query. When $F$ changes, there are two cases. First, assume that $v$ is in on the path $U$. Let $d'$ be the number of edges from $u$ to $v$ and $q'$ be the number of these which are heavy. Hence, $F'$, the amount that $F$ increases by, is $(11q'/4) - d'$. Note that $F' \geq f$ is equivalent to $q' \geq (4f'/11) + (4l'/11)$. We then use this to bound the expected value of $F'$.

$$
\begin{aligned}
E[F'] &= \int_{f\geq 0} f \cdot \Pr[F' = f] \\
&= \int_{f\geq 0} \Pr[F' \geq f] \\
&\leq \int_{f\geq 0} \sum_{m\geq f} \Pr\left[d' = m \text{ and } q' \geq \frac{4f'}{11} + \frac{4m}{11}\right] \\
&= \int_{f\geq 0} \sum_{m\geq f} \Pr\left[d' = m \text{ and } q' \geq \left(1 + \left(\frac{12f}{11m} + \frac{1}{11}\right)\right)\frac{m}{3}\right]
\end{aligned}
$$

If $d'$ is fixed to be $m$, then $q'$ is binomially distributed with mean $m/3$. Using a Chernoff bounds we know that $\Pr[q' \geq (1+\delta)(m/3)] \leq e^{-\delta^2 m/6}$. In our case, $\delta = ((12f/11m) + (1/11))$. Hence,

$$
\begin{aligned}
E[F'] &\leq \int_{f\geq 0} \sum_{m\geq f} \exp\left(-\left(\frac{12f}{11m} + \frac{1}{11}\right)^2 \cdot \frac{m}{6}\right) \\
&= \int_{f\geq 0} \sum_{m\geq f} \exp\left(\frac{-24f^2}{121m}\right) \cdot \exp\left(\frac{-4f}{121}\right) \cdot \exp\left(\frac{-m}{726}\right) \\
&= \sum_{m\geq 0} \exp\left(\frac{-m}{726}\right) \int_{0\leq f\leq m} \exp\left(\frac{-24f^2}{121m}\right) \cdot \exp\left(\frac{-4f}{121}\right) \\
&= \sum_{m\geq 0} \exp\left(\frac{-m}{726}\right) \cdot O\left(\exp\left(\frac{-24}{121m}\right)\right) \\
&= \sum_{m\geq 0} \exp\left(\frac{-m}{726}\right) \cdot O(1) \\
&= O(1)
\end{aligned}
$$

This completes that case when the node $v$ for which $F$ is maximized is in on the path $U$. The only remaining case is when $v$ is a child of a node on the path $U$. For such nodes, $q(v)$ can be at most one more than the value of $F$ on $v$'s parent. Thus, the expected change of $F$ of siblings of nodes in $U$ is at most an additive constant more than the expected change on the nodes in $U$.

Now consider a triangle operation to a node $u$. The protocols learns all the labels to a depth $\gamma$ below $u$. For any node $v$, this increases $q(v)$ by at most $\gamma$. The increase in $d(v)$ has to be at least the increase of $q(v)$. Thus, $F$ can increase by at most $((11\gamma/4) - \gamma) \leq 2\gamma$. □

We are now ready to establish the lower bound for the path and triangle game.

LEMMA 3.3. *Any protocol for the path and triangle game that makes fewer than* $T(n) = \Omega(\log \frac{n}{c} / \log \frac{1}{\epsilon})$ *queries fails with probability at least* $\frac{3}{4}$ *to find a rich path.*

PROOF. Finding a rich path involves finding a leaf $u$ with $q(u) \geq ((4/10)L - \log_2 2c)$, $l(u) = L$, and $F(u) = (11/4)q(u) - l(u) \geq (11/4)((4/10)L - \log_2 2c) - L = ((1/10)L - (11/4)\log_2 2c)$. However, Lemma 3.2 proves that at each time step, the expected change in $F$, is at most $2\gamma + \beta$. Therefore, after fewer than $T(n)$ queries, $E[F]$, the expected value of $F$, is at most $(2\gamma + \beta)T(n)$. By Markov's inequality, the probability that $F \geq 4E[F]$ is at most 1/4. Hence, setting $T(n) = 1/4(2\gamma + \beta)((1/10)L - 11/4 \log_2 2c)$ gives a contradiction. Plugging in $\gamma = 2 + \log_2(1/\epsilon)$ and $\beta = O(1)$ gives $T(n) = \Omega(\log n/c / \log 1/\epsilon)$ as required. □

And finally, we give the lower bound for cake cutting.

THEOREM 3.4. *If a protocol can only make* $1 + \epsilon$ *approximate queries, and c-fairness is required, then the complexity of any randomized protocol for cake cutting is* $\Omega(n \log n/c / \log 1/\epsilon)$.

PROOF. We again use Yao's technique. We assume that each player has a value function specified by an independent random value tree. Let $T(n)$ be defined as in the statement of Lemma 3.3. Because the random value trees are independent, we can conclude as in the proof of Lemma 3.3 that if a player receives fewer than $T(n)$ queries, then he fails to obtain a thin-rich piece with probability 3/4. If the cake cutting protocol makes fewer than $\frac{1}{4}nT(n)$ queries, then this is the case for more than $\frac{3}{4}n$ of the players. Hence, the expected number of players that do not obtain a thin-rich piece is at least $(9/16)n$ and because these events are independent, with high probability more than half the players fail to get a thin-rich piece. We can then conclude as in the proof of Theorem 2.4 that this contradicts the correctness of the protocol. □

We finish with a few comments on the tightness of our lower bounds with approximate queries. If exact queries are replaced by $1+\epsilon$-approximate queries, then the $\Theta(n \log n)$ time divide and conquer protocol returns only $(1+\epsilon)^{\log(n)}$-fair pieces, because the error accumulates multiplicatively at each of the $\log(n)$ levels of recursion. Doing the same for the $\Theta(n^2)$ time protocol introduces only $1+\epsilon$ error to the final fairness. If the model allows only $1+\epsilon$-approximate queries, for some constant $\epsilon$, but requires only $(1+\epsilon)^{\log(n)} \ll n^{1-\delta}$-fairness, then our lower bound of $\Omega(n \log n)$ is tight. If the model allows only $1+1/\log n$-approximate queries and requires $O(1)$-fairness, then our lower bound of $\Omega(n \log n / \log \log n)$ is off by at most a $\log \log n$ term.

## 4. CONCLUSION

The most obvious open question arising from this paper is the randomized complexity to achieve exact fairness with exact queries.

Note that cut queries seem to be more powerful than evaluation queries. For example, the randomized lower from Section 3 holds even if the evaluation queries are allowed to be exact, but breaks down if cut queries are allowed to be exact. An interesting question is whether one can achieve exact fairness deterministically with $O(n)$ exact cut queries and $O(n^2)$ exact evaluation queries. Also while our definition of approximate queries seems to be the most natural one, some further investigation of alternative definitions and their relationship might be interesting.

Subsequent to this work, we obtained a randomized algorithm that achieves approximate fairness for honest players with $O(n)$ queries [Edmonds and Pruhs 2006;

Edmonds et al. 2008]. The outstanding open question in this area is what is the randomized complexity to achieve exact fairness with exact queries.

## REFERENCES

BRAMS, S. AND TAYLOR, A. 1996. *Fair Division—From Cake Cutting to Dispute Resolution*. Cambridge University Press.

EDMONDS, J. AND PRUHS, K. 2006. Balanced allocations of cake. In *Proceedings of the IEEE Symposium on the Foundations of Computer Science*. 623–634.

EDMONDS, J., PRUHS, K., AND SOLANKI, J. 2008. Confidently cutting a cake into approximately fair pieces. In *Proceedings of the International Conference on Algorithmic Aspects in Information and Management*. 155–164.

EVEN, S. AND PAZ, A. 1984. A note on cake cutting. Disc. Appl. Math. 7, 285–296.

KRUMKE, S. O., LIPMANN, M., DE PAEPE, W. E., POENSGEN, D., RAMBAU, J., STOUGIE, L., AND WOEGINGER, G. J. 2002. How to cut a cake almost fairly. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*. 263–264.

MAGDON-ISMAIL, M., BUSCH, C., AND KRISHNAMOORTHY, M. 2003. Cake cutting is not a piece of cake. In *Proceedings of the Symposium on Theoretical Aspects of Computer Science*. Lecture Notes in Computer Science Series, vol. 2607. 596–607.

ROBERTSON, J. AND WEBB, W. 1995. Approximating fair division with a limited number of cuts. *J. Combinat. Theory, Ser. A 72*, 340–344.

ROBERTSON, J. AND WEBB, W. 1998. *Cake-Cutting Algorithms: Be Fair If You Can*. A.K. Peters, Ltd.

SGALL, J. AND WOEGINGER, G. J. 2003. *A lower bound for Cake Cutting*. Lecture Notes in Computer Science Series, vol. 2461. Springer-Verlag.

STEINHAUS, H. 1948. The problem of fair division. *Econometrica 16*, 101–104.

WOEGINGER, G. J. 2002. An approximation scheme for cake division with a linear number of cuts. In *Proceedings of the European Symposium on Algorithms*. Springer-Verlag, 896–901.

Received September 2008; revised September 2009; accepted September 2009