DOI:10.1145/3382129

A Bounded and Envy-Free Cake Cutting Algorithm

By Haris Aziz and Simon Mackenzie

Abstract

We consider the well-studied cake cutting problem in which the goal is to find an envy-free allocation of a divisible resource based on queries from agents. The problem has received attention in mathematics, economics, and computer science. It has been a major open problem whether there exists a discrete and bounded envy-free protocol. We report on our algorithm that resolved the open problem.

1. INTRODUCTION

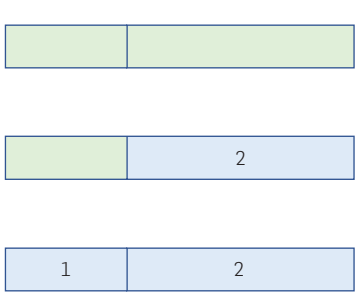
The cake cutting problem is a fundamental mathematical problem in which the 'cake' is a metaphor for a heterogeneous divisible resource represented by the unit interval [0, 1].[4] The resource could represent time, land, or some computational resources. The goal is to allocate the cake among n entities that are referred to as 'agents.' Each agent's allocation consists of a collection of subintervals. Each of the agents is assumed to have additive and nonnegative valuations over segments of the interval. A cake-cutting algorithm/protocol interacts with the agents in order to identify a fair allocation.

One of the most important criteria for fairness is *envy-freeness*. An agent envies another if she would have preferred to receive the other's piece rather than hers. A cake cutting protocol/algorithm is called *envy-free* if each agent is guaranteed to be nonenvious if she reports her real valuations. If a protocol is envy-free, then an honest agent will not be envious even if other agents misreport their valuations.

The interaction of the protocol with the agent uses two types of queries EVALUATE and CUT. EVALUATE asks an agent i to report her value for the subinterval between two points x and y. CUT asks an agent i to choose a point y such that the interval between a given x and y is worth a given value t. This natural query model was popularized by Robertson and Webb.[8]

How does an envy-free protocol look like? This is perhaps best illustrated with the most famous envy-free cake cutting protocol. It is the *Cut and Choose Protocol* for two agents. One agent is asked to divide the cake into two equally preferred pieces. The other agent is then asked to pick her most preferred piece whereas the cutter gets the remaining piece. The protocol is explained pictorially in Figure 1. It is formally specified as Algorithm 1. For a piece of cake D (which is just a subset of the cake), we write $V_i(D)$ to denote the agent i's value for the piece D. The proof that the Cut and Choose Protocol is envy-free is as follows. Agent 1 gets one of the equally preferred pieces so she is not envious. Agent 2 gets the piece that she prefers at least as much as the other piece so she is also not envious.

Figure 1. Cut and choose protocol.



Algorithm 1. Cut and choose protocol.

Input Cake $R = [0, 1]$ and two agents 1 and 2.
Output An envy-free allocation of R.

1: Ask agent 1 for her value $V_1(R)$. Then ask agent 1 to place a mark x on the cake so that $V_1(0, x) = V_1(x, 1)$. Divide the cake into two pieces $[0, x]$ and $[x, 1]$.
2: Ask agent 2 for her value $V_2(0, x)$ and $V_2(x, 1)$. If $V_2(0, x) \geq V_2(x, 1)$, give agent 2 piece $[0, x]$. Otherwise, give piece $[x, 1]$ to agent 2.
3: Give agent 1 the remaining piece.

Is there a cake cutting algorithm that is envy-free for three, four, or more number of agents? The question has been the topic of intense study in the past decades. It dates back to the work of mathematician Hugo Steinhaus who presented the cake cutting problem in the 1940s.[8,9] For an enjoyable overview of the history of the cake cutting problem, we refer to the Communications of the ACM paper by Procaccia[7] the popular book by Brams and Taylor.[4] For the case of three agents, an elegant protocol was independently discovered by John L. Selfridge and John H. Conway around 1960. Before our work, a general envy-free cake cutting algorithm using a finite number of steps and cuts was proposed by Brams and Taylor.[3] However, it can require an arbitrarily large number of steps, even for four agents. This led to the question of whether there exists a bounded envy-free algorithm. In other words, does there exist an envy-free algorithm that has a provable bound on the number of steps which is only dependent on a function of n (the number of agents)?

In this paper, we report on the first bounded and envy-free cake cutting algorithms.[1,2] Next, we present the ideas behind the general algorithm.

The original version of this paper, entitled "A Discrete and Bounded Envy-Free Cake Cutting Protocol for Any Number of Agents," was published in the *Proceedings of the 57th Symposium on Foundations of Computer Science*, (2016), 416–427.

2. THE PROTOCOL: AN OVERVIEW

At a high level, our protocol (which is referred to as the Main Protocol) allocates a large enough portion of the cake in an envy-free manner. After that, it tries to add some small portions of the unallocated cake to the allocated part in a structured and envy-free manner with the goal to reduce the problem to envy-free allocation for a smaller number of agents. Throughout the protocol, there is a partial allocation of the cake that is maintained to be envy-free. By partial we mean that the whole cake may not be allocated.

The Main Protocol makes calls to other protocols (in particular the Core Protocol, Discrepancy Protocol, and the GoLeft Protocol) in order to find an envy-free allocation. The Core Protocol is used to obtain an envy-free partial allocation. The Main Protocol applies it many times on the unallocated cake to make the unallocated cake smaller and smaller.

After finding a large enough envy-free partial allocation, the Main Protocol uses two possible ways to decompose our problem into one involving a smaller number of agents. The first case is when we find a situation where some agents are mainly interested in one part of the unallocated cake and other agents are mainly interested in the remaining part. This discrepancy in valuations of the agents is exploited by the Discrepancy Protocol. If the first case does not arise, we use the GoLeft Protocol to exchange suballocations of agents to enable one set of agents to *"dominate"* the other agents. The dominating agents think they will not be envious of the dominated agents even if one of the dominated agents gets all the unallocated cake. In that case, we reduce our problem to that involving the remaining cake and the dominated agents. Domination is a key idea on which our protocol is based and which helps us reduce our problem to a smaller problem. See Figure 2 for an overview of the Main Protocol.

Figure 3 presents a realizable sequence of steps that capture some of the key ideas of our protocol.

3. THE PROTOCOL: MORE DETAILS

In this section, we give more details of each of the components of the Main Protocol.

3.1. Core protocol

A crucial building block of our protocol is the Core Protocol which finds a partial allocation that is envy-free.

Figure 2. A bird's-eye view of our protocol.

Main Protocol

Goal: Find an envy-free allocation of the whole cake.

1. Call the **Core Protocol** (that finds an envy-free partial allocation) several times to get a larger and larger envy-free allocation.
2. Decompose the problem into one with a smaller number of agents via two possible ways:

a) Call the **Discrepancy Protocol** (that exploits how agents value different parts of the unallocated cake): we get two smaller subproblems.

b) If there is no discrepancy, call the **GoLeft Protocol** (implements exchanges of some pieces to enable one set of agents to dominate the other agents). We get one smaller subproblem (with less number of agents).

The Core Protocol asks one of the n agents—the "cutter"—to divide the cake into n equally preferred pieces. Recall that this step is similar to the first step of the Cut and Choose Protocol. It then finds a possibly partial allocation in which each agent's allocation is a contiguous piece of the cake. Each agent receives one of the pieces defined by the "cutter". The agents may get the pieces in trimmed form. We guarantee the cutter as well as at least one other agent to get a full piece, and that no agent envies another agent. Another feature of the allocation is that for each piece that is partially allocated, the exact point at which it has been cut off corresponds to the mark by another agent to ensure she is not envious of that piece. When we first designed the Core Protocol, it was designed to establish the existence of an allocation that satisfies the properties discussed above. Once the existence of such an allocation is established, there is a simpler way to define a protocol which achieves such an allocation. The general idea for the simplified version was made explicit in an interesting and detailed follow-up paper, which solved the sister problem for the case of chores or burnt cake (agents have nonpositive valuations).[5] Here we present a simplified version of the Core Protocol (Algorithm 2) for cake cutting. The protocol requires at most $(n!)^2 n$ queries.

Algorithm 2. (Simplified) core protocol.

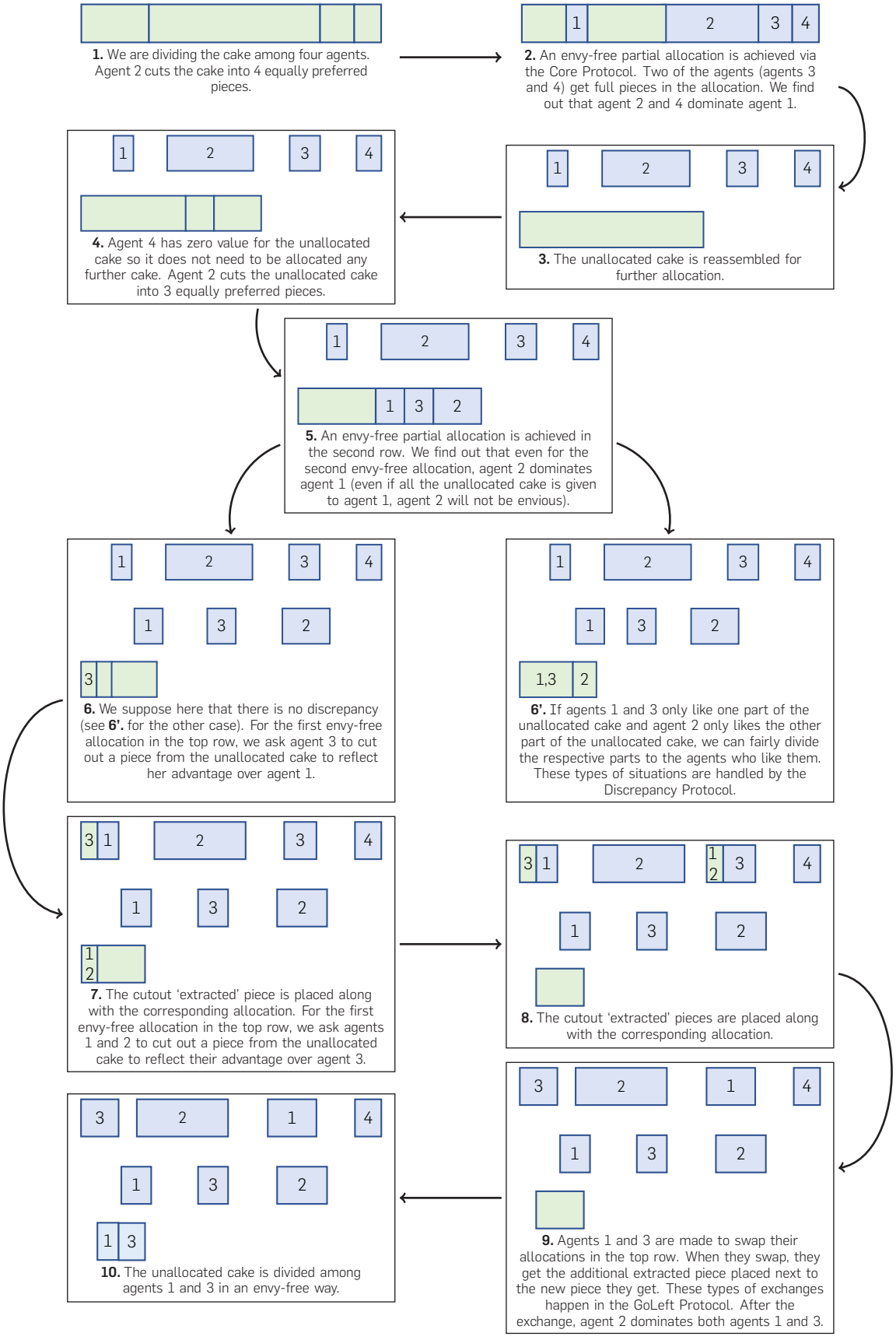
Input Agent set N, a cutter $i \in N$ and cake R.
Output An envy-free allocation of cake $R' \subseteq R$ for agents in N and an updated unallocated cake $R \setminus R'$.

1: Ask agent i to cut the cake R into n equally preferred pieces $(p_1,\ldots,p_n)$.
2: **for** each permutation $(a'_2,\ldots,a'_n)$ of $N\setminus\{i\}$ **do**
3: **for** each permutation $(p'_1,\ldots,p'_n)$ of the n pieces $(p_1,\ldots,p_n)$ **do**
4: Give p'_1 to i
5: **for** $j = 2$ to n **do**
6: Give p'_j to a'_j. Ask a'_j to trim any of the pieces $p'_{j+1},\ldots,p'_n$ if needed so the value of the pieces does not exceed the value a'_j has for her allocation p'_j.
7: **if** the allocation p corresponding to the permutation of agents and pieces is envy-free **then**
8: **return** the allocation p (which is called a Core allocation)
9: **else**
10: Reattach the trimmed parts to regain the original pieces.

In the Core Protocol, the cutter agent gets a full piece. Another agent also gets a full piece. So from the cutter's perspective, at least $2/n$ of the cake is allocated by one call of the Core Protocol. Equivalently, the cutter thinks that her value of the remaining cake is at most $(n-2)/n$ of her value of the full cake.

If we call the Core Protocol with a different cutter each time to further allocate the unallocated cake, we just need n calls of the Core Protocol to obtain an envy-free partial allocation which also satisfies proportionality (gives each agent value at least $1/n$ of the whole cake). Algorithm 3 does exactly that and in $n!^2 n^2$ queries finds a partial allocation that

Figure 3. Illustration of some of the ideas of the protocol. The terminal states are 6' and 10.



satisfies envy-freeness and proportionality—two of the most important fairness concepts.[a] The remainder of the paper describes what to do when we do want to allocate the whole cake.

Algorithm 3. An envy-free and proportional protocol.

Input Agent set N and cake R.
Output An envy-free and proportional allocation of the cake that may not allocate the whole cake.
1: **for** $i = 1$ to n **do**
2: **if** there is unallocated cake, **then** run the Core Protocol on the unallocated cake with i as the specified cutter.
3: **return** the allocation.

3.2. Domination and significant advantage

As the Core Protocol by itself is not powerful enough to allocate all the cake in bounded time, we rely on the idea of *domination* with the goal to decompose our problem into one involving a fewer number of agents. In this section, we denote an agent i's allocation by X_i.

Recall that in an envy-free allocation, each agent i thinks she has an advantage (even if it is zero advantage) over each other agent j:

$$V_i(X_i) - V_i(X_j) \geq 0.$$

Domination is an extreme form of advantage. An agent i *dominates* another agent j if she is not envious of j even if the unallocated cake R is given to j:

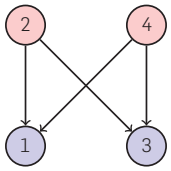
$$V_i(X_i) - V_i(X_j) \geq V_i(R).$$

The other protocols are used with the following objective in mind: find a set of agents $A \subset N$ such that each agent in $N \setminus A$ dominates each agent in A. In order to ensure that each agent in some set $N \setminus A$ dominates each agent in A, it requires changing the current allocations of the agents as well as the unallocated cake. While doing these changes, we ensure that the current partial allocation remains envy-free. By identifying such a set $N \setminus A$, we reduce the problem to envy-free allocation for a smaller number of agents. The agents in $N \setminus A$ are not envious whatever the unallocated cake is allocated among agents in A. This crucial idea is illustrated in Figure 4.

Dominance of an agent i over another agent j has a close relation with agent i considering herself as having a 'significant advantage' over j. In order to define significance, we consider a suitable large constant bounded by some function over n. For a partially allocated cake, and piece a, an agent i finds value $V_i(a)$ significant if the value is at least $V_i(R)\left(\frac{n-2}{n}\right)$ where R is the unallocated cake

Significance of a piece is with respect to the residue, so if the residue becomes smaller, a significant value remains significant. The rationale for defining a significant value is that if an agent i thinks she has a significantly higher value for her allocation than she has for agent j's allocation, then this significant advantage can be changed into domination

Figure 4. In the figure, an agent points to another agent if the former dominates the latter. Suppose we find an envy-free partial allocation among four agents such that each agent in {2, 4} dominates each agent in {1, 3}. Then we can simply allocate the remaining cake among agents in {1, 3} in an envy-free way.



by calling the Core Protocol a bounded number of times with i as the cutter. We explain this idea below.

Suppose we partially allocate the cake and agent i gets allocation X_i whereas agent j gets allocation X_j. Suppose that agent i thinks she has a significant advantage over agent j:

$$V_i(X_i) - V_i(X_j) \geq V_i(R)\left(\frac{n-2}{n}\right)^B.$$

Consider the situation where we run the Core Protocol over the unallocated cake R with agent i as the specified cutter and we do it B times so that the eventual unallocated cake is R^*. Then

$$V_i(R^*) \leq \left(\frac{n-2}{n}\right)^B V_i(R).$$

Thus, after B calls of the Core Protocol, agent i who previously had a significant advantage over agent j now dominates her:

$$V_i(X_i) - V_i(X_j) \geq V_i(R)\left(\frac{n-2}{n}\right)^B \geq V_i(R^*).$$

When we get a Core Protocol outcome, the cutter already has a significant advantage over the agent who got the least cake in the cutter's estimation. This significant advantage can easily be converted into domination by calling the Core Protocol. The main challenge is to obtain domination relations between more pairs of agents. Throughout the main protocol, the tentative partial allocation remains envy-free. Secondly, if an agent dominates another agent, the domination is maintained despite updates to the allocation.

3.3. Extraction

After we have called the Core Protocol on the updated unallocated cake, a sufficient but bounded number of times, we are in a position to *extract* from the residue. In each of the calls of the Core Protocol, there was a corresponding envy-free allocation. By envy-freeness, in each such allocation, each agent j has a nonnegative advantage over another agent i. For each of the Core allocations and for each $i, j \in N$, agent i is asked to extract a piece from the unallocated cake of value of the advantage over j in that Core allocation.

The extracted piece e will be in consideration to be attached to i's corresponding allocated piece so that j is indifferent between her allocation and i's allocation. If j's intended extraction has a significant value, we do not extract because we only want to extract pieces from the remainder which are not significant for all the agents. If the intended extraction is not significant, we put it on a side for

[a] Note that finding an envy-free allocation that may be partial is a trivial problem: allocate nothing!

consideration for attachment. If it cannot be made unanimously insignificant, then we say that the piece is discrepant and we call the Discrepancy protocol which either exploits or 'eliminates' this discrepancy.

Figure 5 shows how agents extract pieces from the unallocated cake R. In the figure, we consider extractions by agents 2, 3, and 4 based on their advantage over agent 1. Agent 2 thinks that her advantage over agent 1 is of the same value as her value for the leftmost extracted piece. Agent 4 thinks that her advantage over agent 1 is of the same value as the sum of her values for two leftmost extracted pieces.

The extracted pieces will be attempted to be attached to agent 1's piece as indicated in Figure 6.

Suppose we have a set of Core Protocol allocations and the corresponding extracted pieces placed in the appropriate order. We call a set of Core Protocol allocations *isomorphic* to each other if for each piece c_i in agent i's allocation, the agents who extracted cake from the residue and associated to c_i are the same and did so in the same order. Later, we will identify a subset of Core Protocol allocations that are isomorphic to each other. Isomorphic allocations will be considered later by the GoLeft Protocol.

3.4. Discrepancy protocol

When pieces are being extracted from the residue, it may be the case that one of the pieces e in consideration for extraction is significant for some agent. In that case, the piece is not extracted and the Discrepancy Protocol is called that either eliminates or exploits this discrepancy. The discrepant piece e is kept aside from the residue. If the piece is "almost significant," we can make it significant by reducing the residue by calling the Core Protocol a bounded number of times.

By doing this, either the discrepant piece becomes unanimously significant or we still have the case that some agents consider e significant and others do not. The first case is helpful because there is no discrepancy in terms of significance and our protocol makes use of this consistency. In the second case, if there exists some $i \in N$ such that $V_i(R)/n < V_i(e) < V_i(R)n$, we continue running the Core Protocol with agent i as the cutter. By doing so, we achieve in a bounded number of calls of the Core Protocol the situation where for each agent i, either $V_i(e) \geq V_i(R)n$ or $V_i(e) \leq V_i(R)/n$. This situation is explained in Figure 7.

Figure 5. Agents extracting pieces from the remaining cake up to their advantage over agent 1.

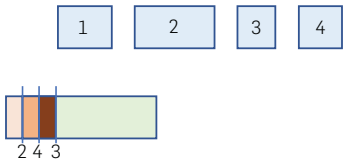
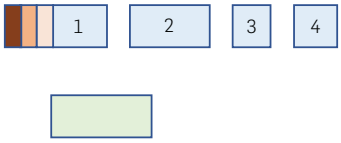


Figure 6. Extracted pieces placed next to agent 1's allocation for the purpose of attachment.



Algorithm 4. Main protocol—high-level sketch.

Input: Cake R and a set of agents N.
Output: An envy-free allocation.

1: **Core Allocations**: generate core allocations by repeatedly dividing the unallocated cake via the Core Protocol a bounded number of times.
2: **Extraction**: extract pieces from the residue corresponding to the core allocation pieces as long as the pieces are not significant for any agent as explained in Section 3.3. While extracting pieces, if some piece a is significant for at least one agent, call the Discrepancy Protocol as explained in Section 3.4. It ensures that now either all agents consider the piece significant (in which case it is not attached) or we decompose the main problem into two subproblems for the Main Protocol where some agents are to be given a and the others are to be given the remaining unallocated cake.
3: **GoLeft**: Call the GoLeft Protocol to attach extracted pieces to the corresponding Core allocations. The GoLeft protocol returns a subset of agents $A \subset N$ such that each agent in $N \setminus A$ dominates each agent in A. The central idea of GoLeft is to facilitate exchanges of suballocations of agents.
4: Call the Main Protocol to allocate the remaining cake to agents in A.
5: **return** allocation of the cake to the agents.

If $V_i(e) \geq V_i(R)n$, then i is predominantly interested in e rather than the residue. If $V_i(e) \leq V_i(R)/n$, then i is predominantly interested in R. Because the piece that agents are predominantly interested in has n times more value than the other piece, any agent who gets an envy-free (and hence proportional) allocation of the preferred piece also gets at last $1/n$ value of the preferred piece. The value is at least as much as the value of the piece that is less preferred.

3.5. Main protocol

Continuing to call the Core Protocol on the updated remaining cake gives no guarantee that the cake will be allocated fully even in finite time. Hence, we need to use other protocols which are called by the Main Protocol. We gave an intuitive idea of the Main Protocol in Figure 2. We give a more detailed high-level sketch of the protocol in the form of Algorithm 4.

The first two stages of the Main Protocol are making calls to the Core Protocol to further allocate the cake and then to

Figure 7. Discrepancy. Agents in A think that the left part has n times more value than the right part. Agents in $N \setminus A$ think that the right part has n times more value than the left part. In that case, if we allocate the left part to A in an envy-free way and the right part to $N \setminus A$ in an envy-free way, we obtain an overall envy-free allocation for N.

implement the extraction as explained in the previous sections. While pieces are being extracted, we may have to call the Discrepancy Protocol. Throughout the steps of the Main Protocol, we maintain an envy-free allocation as well as keep track of the updated unallocated cake. After that, the Main Protocol calls the GoLeft Protocol. In the subsequent section, we give further details of the GoLeft Protocol.

3.6. GoLeft protocol

In this section, we give an overview of the GoLeft Protocol (Algorithm 5). When the GoLeft Protocol is called, we already have a bounded number of envy-free allocations due to the calls to the Core Protocol. We also have extracted pieces from the residue that will be considered for attachments to the corresponding Core allocations of the agents.

The purpose of extracting pieces from the residue is that we can attach them to the corresponding Core allocation piece of i so that j is indifferent between her allocated piece and i's piece. This makes it easier for j to switch one of her pieces if she gets the additional insignificant extraction. Making agents exchange their allocations while additionally giving them additional extracted pieces is useful to diversify the relations of agents having a significant advantage over others.

We elaborate on why attachment is helpful to obtain additional significant advantages. Let us say that in a number of Core allocations, agent k has a significant advantage over agent i's allocation and agent j has an insignificant advantage over i's allocation. In order for k to have a significant advantage over j rather than i, we want to make some local envy-free preserving operations so that j gets i's allocated piece along with j's insignificant extraction corresponding to j's advantage over that piece of i's.

Permutation graph. When the GoLeft Protocol starts, it first identifies a working set S of C Core allocations from out of the C' Core allocations that we focus on. As C' is chosen to be large enough, we can find C Core allocations that are isomorphic. The protocol then constructs a *permutation graph* corresponding to the working set of isomorphic allocations.

In the permutation graph, each node i corresponds to an agent i who holds a set of isomorphic pieces along with her attached extracted pieces in the working set of isomorphic allocations S. Agent i points to agent j if j holds isomorphic pieces in S that have had all attachments up till i's extracted pieces. Each node has an indegree of one. Initially, the permutation graph consists of all nodes having self-loops (see Figure 8).

We divide the nodes of the permutation graph into sets T and T'. Set T is the set of nodes/agents such that the isomorphic pieces held by them in S have not had $n-1$ attachments). T' is the set of nodes/agents such that the isomorphic pieces held by them in S have had $n-1$ attachments.

The protocol identifies a cycle in the permutation graph that includes at least one node i from T. Such a cycle always exists. In each of the working set S of isomorphic allocations, we implement an exchange of pieces held by agents in the cycle: each agent in the cycle is given the piece corresponding to the node that the agent points to in the cycle. After implementing the exchange, the permutation graph is updated to reflect the exchange. In the exchange, if an agent gets an inferior piece, she always gets the additional extracted pieces associated with it up till the agent's extractions. Hence, each agent's value from her allocation is preserved in each allocation in S even if she gets a different piece than in the original Core allocation. For any agent i, as long as no agent gets extracted pieces beyond i's extraction, i will not be envious. In the GoLeft protocol, it can be the case that some agent j gets extracted pieces beyond i's extracted pieces but before any such attachments in the last part of the GoLeft protocol, we ensure that no envy arises.

After implementing the cycle, we focus on a node $i \in T$ that was in the cycle. For agent/node i, we know that for all allocations in the working set S, agent i has been allocated the original isomorphic pieces c_k as well as all associated pieces up till i's extracted piece. If the piece of cake agent i is currently allocated in the allocations S has no more extracted pieces left to attach to it, but it has not had $n-1$ attachments, this means that all agents who have not had their corresponding piece extracted/attached have a significant advantage over agents who have had an extracted piece attached. In this case, the GoLeft Protocol returns the set of dominated agents to the Main Protocol and we are left with a smaller envy-free allocation problem because it involves a fewer number of agents.

In case node i does not lead to an exit from the GoLeft Protocol, we know that there are associated pieces that can still be attached to the isomorphic pieces held by i in the working set of Core allocations S. We focus on the next set of associated pieces $e_{k(l+1)}$ that we are interested to attach to the pieces c_k that have already had associated pieces $e_{k2}, e_{k3}, \ldots, e_{kl}$ attached in their corresponding main pieces c_k (see Figure 9). Additionally attaching pieces $e_{k(l+1)}$ to pieces c_k is useful in making the agent who extracted them interested in the pieces c_k because of the additional $e_{k(l+1)}$ as well as the previous attachments.

Avoiding envy when attaching extracted pieces. Naively attaching the pieces can be problematic and spoil the envy-freeness of the allocation that we maintain. We deal with the issue as follows.

- The agents who did not extract pieces associated with the c_k pieces as well as agents who extracted pieces that have not been attached are asked to 'reserve' a big enough subset $S' \subset S$ of allocations in which they value the difference between their bonus value for c_k and the extracted pieces currently attached to c_k the most. These allocations S' are removed from S and their remaining unattached associated pieces are sent back to the residue. By maintaining the advantages in the Core alloca-

Figure 8. Initial state of the permutation graph along with the corresponding state of an allocation representative of the working set of isomorphic allocations.

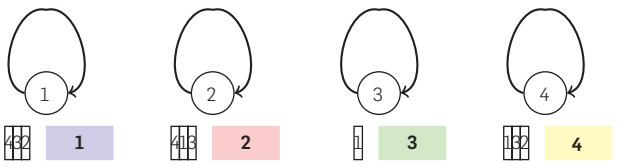
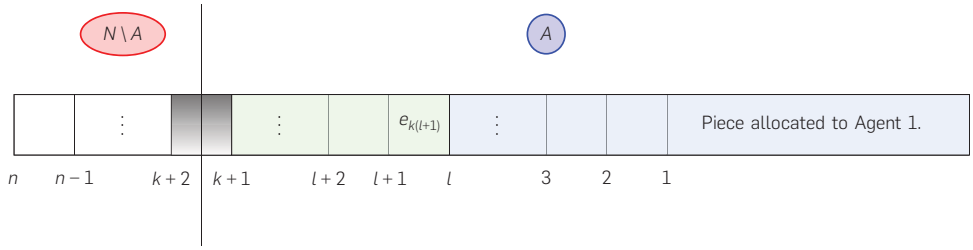


Figure 9. Illustration of the GoLeft protocol on a particular piece of cake that is originally allocated to agent 1. Agents $k+2$ to n will not go left and are the prospective dominators because they find the shaded space between the trims of $k+2$ and $k+1$ significant. Agents 2 to $k+1$ are the agents that go left.



tions S', such agents will not be envious even if some agent in $\{1, ..., l\}$ additionally gets all other extracted pieces $e_{k(l+1)}$ in the remaining Core allocations in S.

Algorithm 5. GoLeft protocol—high-level sketch.

Input: Set of C' allocations, extracted pieces corresponding to the C' Core allocations, and residue R.
Output: A set of agents $A \subset N$ such that agents in $N \setminus A$ dominate agents in A.

1: Select C isomorphic allocations (set S); Build the permutation graph.
   - T, the set of nodes with pieces for which $n-1$ extracted pieces have not been attached.
   - T', the set of nodes with pieces for which $n-1$ extracted pieces have been attached.
2: **while** there is a node in T **do**
3: Find a cycle that includes a node that is from T (such a cycle always exists).
4: In the cycle identified, let each agent in the cycle get the allocation she points to up till her extractions.
5: **if** there is a piece p corresponding that is not from T' but has no more associated pieces to be attached **then**
6: Consider the set of agents A who either owned the original piece p or whose extracted pieces have already been attached to p. Return the dominated set of agents A.
7: **Attachment:** consider the set of isomorphic Core allocation pieces $\{c_k\}$ that have already had associated pieces $\{e_{k2}\}$, $\{e_{k_3}\}$, ..., $\{e_{kl}\}$ attached to them but some extracted pieces have not been attached. Attach in a subset of the allocations in C' the set of extracted pieces $\{e_{k(l+1)}\}$ to the set of pieces $\{c_k\}$, thus making $\{c_k\}$ desirable to the agent who extracted $\{e_{k(l+1)}\}$. In order to attach the pieces without creating envy, a subset S', $S'' \subset S$ of Core allocations is removed from the working S of Core allocations. The Core allocations in $S' \cup S''$ do not undergo attachments or further changes. Update the permutation graph to reflect the attachment. If the piece has had all $n-1$ extracted pieces attached, add the corresponding node to T' and make every node point to it.

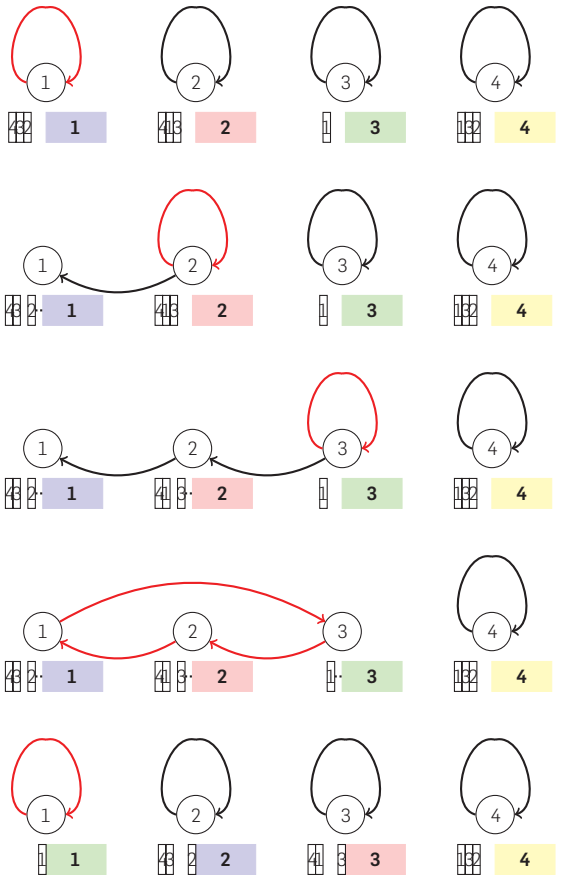
- The agents indexed from 1 to l who have all already had their extracted pieces attached to c_k are asked to choose a high enough fraction of the Core allocations in S in which they value the $e_{k(l+1)}$ pieces. We call these allocations S''. The $e_{k(l+1)}$ pieces from S'' are bunched together and the Main Protocol is called to divide this cake in an envy-free way among the agents indexed from 1 to l where l is strictly less than n. As envy-freeness implies proportionality, they derive enough value that they will not be envious if the agent indexed $l+1$ gets all other pieces in set $e_{k(l+1)}$. The corresponding set of allocations S'' is then removed from consideration for updates.

Hence, each time we attach isomorphic extracted pieces $e_{k(l+1)}$ to isomorphic pieces c_k, we 'freeze' allocations $S' \cup S''$ from the working set S and still maintain an envy-free allocation. Note that in the allocations that remain in S, agents may currently hold a different isomorphic piece than they previously did, but as they also hold the corresponding attachments associated with the isomorphic piece, each agent's total value in each isomorphic allocation in S stays the same. In Figure 10, we show the states of the permutation graph and the corresponding representative Core allocation as well as the corresponding extracted pieces.

When the protocol attaches extracted pieces $e_{k(l+1)}$ to allocated pieces c_k currently held by agent l, it deletes the incoming edge of node/agent l and replaces it by an edge coming from agent $l+1$ who extracted pieces in $e_{k(l+1)}$. Intuitively, $l+1$ is now willing to be allocated to c and its attached pieces instead of her current pieces in S. We delete previous edges to ensure that until termination, nodes in T have an indegree of exactly 1, which guarantees that no matter the cycle involving a node in T found by the protocol, we will make progress towards termination. The following example shows how progress is made in attaching extracted pieces to the working set of isomorphic Core allocations.

Example 1. *In Figure 10, we demonstrate how the permutation graph along with the working set of isomorphic allocations changes in the GoLeft Protocol. Note that even when the representative allocation changes, there still exist allocations isomorphic to the previous representative allocations but these allocations have been removed from consideration from the working set of allocations. The colored/shaded pieces represent the pieces given by the Core Protocol to each agent. The small pieces on the left of the colored pieces are extracted pieces, each labeled by the agent who extracted it. At first, the extracted pieces are associated with a specific allocated piece. Then they are attached to it*

Figure 10. Permutation graph along with the corresponding state of an allocation representative of the working set of isomorphic allocations.



(represented by the dotted lines). Finally, when a colored/shaded piece is real-located to a new agent, the extracted pieces attached to it are also allocated to the new agent (in the diagram we now aggregate the extracted piece to the main piece). In the second state of the isomorphic allocation, agent 2 *points to agent* 1 *because the piece extracted by agent* 2 *has been attached to* 1*'s held piece. In the third state of the isomorphic allocation, agent* 3 *points to agent* 2 *because the piece extracted by agent* 3 *has been attached to* 2*'s held piece. In the fourth state of the isomorphic allocation, agent* 1 *points to agent* 3 *because the piece extracted by agent* 1 *has been attached to* 3*'s held piece. In the fifth state, the agents* 1, 2, and 3 *exchange their currently held piece and are allocated cake up to their extracted piece. In the fifth (last) state of the isomorphic allocation, agent* 1 *holds a piece up till her extraction but neither agent* 2 *or* 4 *extracted pieces for the piece that agent* 1 *holds. This means that agents* 2 *and* 4 *have a significant advantage over agent* 1*. Initially, the piece was held by* 3 *and still is in discarded isomorphic allocations. This implies a significant advantage of* 2 *and* 4 *over* 3*. Therefore, agent* 2 *and* 4 *can be made to dominate* 1 *and* 3.

By attaching enough extracted pieces in the appropriate order, the GoLeft Protocol finally arrives at a point where there is some isomorphic set of pieces c_k in the set S for which all possible associated pieces have been attached but there is some set of agents $N \setminus A$ who do not have associated pieces. The reason agents in $N \setminus A$ could not extract such pieces is because they had a unanimous significant advantage over the agent indexed 1 who got the pieces c_k. By gradually attaching (unanimously insignificant) associated piece to pieces c_k and ensuring that all agents who did extract the corresponding pieces do get some isomorphic piece in c_k (along with the associated insignificant attachments), we make sure that agents in $N \setminus A$ now dominate agents in A. At this point, we can return from the GoLeft Protocol. We have successfully reduced our envy-free allocation problem to that involving less number of agents. By recursively calling the Main Protocol to allocate the remaining cake to agents in the smaller set A, we eventually allocate the whole cake.

4. CONCLUSION

We presented a high-level overview of our bounded envy-free protocol. The protocol has an upper bound that is a power tower of six n's. In the other direction, any envy-free protocol requires at least $\Omega(n^2)$ queries.[6]

We additionally show that even if we do not run our protocol to completion, it can find in at most n calls of the Core Protocol a partial allocation of the cake that achieves proportionality (each agent gets at least $1/n$ of the value of the whole cake) and envy-freeness. If we allow for partial allocations, an interesting open problem is the following one: can envy-freeness and proportionality be achieved in a polynomial number of steps?

Acknowledgments

Haris Aziz is supported by a UNSW Scientia Fellowship. He thanks Xin Huang, Sven Koenig, Omer Lev, Bo Li, and Simon Rey for helpful feedback. He also thanks Simon Rey for his help in making some of the figures.

References

1. Aziz, H., Mackenzie, S. A discrete and bounded envy-free cake cutting protocol for four agents. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)* (ACM Press, 2016), 454–464
2. Aziz, H., Mackenzie, S. A discrete and bounded envy-free cake cutting protocol for any number of agents. In *Proceedings of the 57th Symposium on Foundations of Computer Science (FOCS)* (2016), 416–427
3. Brams, S.J., Taylor, A.D. An envy-free cake division protocol. *Am. Math. Month 1*. 102 (1995), 9–18.
4. Brams, S.J., Taylor, A.D. *Fair Division: From Cake-Cutting to Dispute Resolution*. Cambridge University Press, 1996.
5. Dehghani, S., Farhadi, A., Taghi Hajiaghayi, M., Yami, H. Envy-free chore division for an arbitrary number of agents. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018* (New Orleans, LA, USA, January 7–10, 2018), 2564–2583.
6. Procaccia, A.D. Thou shalt covet thy neighbor's cake. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)* (AAAI Press, 2009), 239–244.
7. Procaccia, A.D. Cake cutting: Not just child's play. *Commun. ACM 7*, 56 (2013), 78–87.
8. Robertson, J.M., Webb, W.A. *Cake Cutting Algorithms: Be Fair If You Can*. A. K. Peters, 1998.
9. Steinhaus, H. The problem of fair division. *Econometrica*, 16 (1948), 101–104.

Haris Aziz (haris.aziz@unsw.edu.au), UNSW Sydney, Australia; CSIRO Data61, Sydney, Australia.

Simon Mackenzie (simon.mackenzie@data61.csiro.au), CSIRO Data61, Sydney, Australia.