An Improved Envy-Free Cake Cutting Protocol for Four Agents

Georgios Amanatidis[1], George Christodoulou[2], John Fearnley[2], Evangelos Markakis[3(✉)], Christos-Alexandros Psomas[4], and Eftychia Vakaliou[3]

[1] Centrum Wiskunde & Informatica (CWI), Amsterdam, Netherlands
[2] University of Liverpool, Liverpool, UK
[3] Athens University of Economics and Business, Athens, Greece
markakis@aueb.gr
[4] Carnegie Mellon University, Pittsburgh, USA

Abstract. We consider the classic cake-cutting problem of producing envy-free allocations, restricted to the case of four agents. The problem asks for a partition of the cake to four agents, so that every agent finds her piece at least as valuable as every other agent's piece. The problem has had an interesting history so far. Although the case of three agents is solvable with less than 15 queries, for four agents no bounded procedure was known until the recent breakthroughs of Aziz and Mackenzie [2,3]. The main drawback of these new algorithms, however, is that they are quite complicated and with a very high query complexity. With four agents, the number of queries required is close to 600. In this work we provide an improved algorithm for four agents, which reduces the current complexity by a factor of 3.4. Our algorithm builds on the approach of [3] by incorporating new insights and simplifying several steps. Overall, this yields an easier to grasp procedure with lower complexity.

1 Introduction

Producing an envy-free allocation of an infinitely divisible resource is a classic problem in fair division. As it is customary in the literature, the resource is represented by the interval $[0, 1]$, and each agent has a probability measure encoding her preferences over subsets of $[0, 1]$. The goal is to divide the entire interval among the agents so that no one envies the subset received by another agent. We note that the partition does not need to consist of contiguous pieces; the piece of an agent may be any finite collection of subintervals.

The problem has a long and intriguing history. It has been long known that envy-free allocations exist for any number of agents, via non-constructive proofs [8,16,18]. For algorithmic results, the standard approach is to assume access to the valuation functions via *evaluation* and *cut queries* (see Sect. 2). Under this model, we are interested in counting the number of queries needed for producing an envy-free allocation. For two agents, the famous cut-and-choose protocol requires only two queries. For three agents, the procedure of Selfridge and Conway [6] guarantees an envy-free allocation after at most 14 queries. For four

X. Deng (Ed.): SAGT 2018, LNCS 11059, pp. 87–99, 2018.
https://doi.org/10.1007/978-3-319-99660-8_9

agents and onwards, however, the picture changes drastically. The first finite, yet unbounded, algorithm was proposed by [5]. This was followed up by other more intuitive algorithms, which are also unbounded, e.g., [10,13]. Finding a bounded algorithm was open for decades and positive results had been known only for certain special cases, like piece-wise uniform or polynomial valuations [4,7,9]. It was only recently that a major breakthrough was achieved by Aziz and Mackenzie, presenting the first bounded algorithms, initially for four agents [3], and later for an arbitrary number of agents [2].

Despite these significant advances, the algorithms of [2,3] are still of very high complexity. For an arbitrary number of agents, n, the currently known upper bound involves a tower of exponents of n, and even for the case of four agents, the known algorithm requires close to 600 queries. On top of that, these algorithms are rather complicated and their proof of correctness requires tedious case analysis in certain steps. Hence, a clean-cut and more intuitive algorithm is still missing.

Contribution: We focus on the case of four agents and present an improved algorithm that reduces the query complexity roughly by a factor of 3.4 (requiring 61 cut queries and 110 evaluation queries). Our algorithm utilizes building blocks that are similar to the ones used by [3], but by incorporating new insights and simplifying several steps, we obtain a solution with significantly fewer queries. The main differences between our work and [3] are highlighted at the end of this section. Our algorithm works by maintaining a partial allocation along with a leftover residue. Throughout its execution, it keeps updating the allocation and reducing the residue, until certain structural properties are satisfied. These properties involve the notion of *domination*, where we say that an agent i dominates another agent j, if allocating the whole remaining residue to j will not create any envy for i. A crucial part of the algorithm is to get a partial allocation where one agent is dominated by two others. Once we establish this, we then exhibit how to produce a complete allocation of the cake without introducing any envy. Overall, this results in an algorithm with markedly lower query complexity.

Further related work: We refer the reader to the book chapter [12] for a more proper treatment of the related literature. Towards simplifying the algorithm of Aziz and Mackenzie [3], the work of Segal-Halevi et al. [15] (see their Appendix B) proposes a conceptually simpler framework, without, however, improving the query complexity. Apart from the algorithmic results mentioned above, there has also been a line of work on lower bounds. For envy-freeness, Stromquist [17] showed that there is no finite protocol for producing envy-free allocations where all the pieces are contiguous. Later on, Procaccia [11] established an $\Omega(n^2)$ lower bound for producing non-contiguous envy-free allocations. Apparently, there is still a huge gap between the known lower and upper bounds for any $n \geq 4$.

An Overview of the Algorithm. We start with a high level description of the main ideas. As with most other algorithms, our algorithm maintains throughout its execution a partial allocation of the cake, along with an unallocated residue.

The goal is to keep updating the allocation and diminishing the residue, with the invariant that the current partial allocation is always envy-free. Once the residue is eliminated, we are left with a complete envy-free allocation. As mentioned earlier, the notion of *domination* is pivotal in our approach. The algorithm creates certain domination patterns between the agents, working in phases as follows:

Phase One. We find this first phase of particular importance, as it is also the most computationally demanding one. Here the goal is to get a partial envy-free allocation in which some agent is dominated by two other agents as in Fig. 1a. In order to establish dominations among agents, we use as a subroutine the so-called CORE protocol. In the CORE protocol one agent has the role of the "cutter", and the output is a new allocation with a strictly diminished residue. The properties of CORE have several interesting and crucial consequences. First, if CORE is executed twice with the same agent as the cutter, then this cutter dominates at least one other agent in the resulting allocation. Moreover, if we run CORE two more times, we may not get any extra dominations right away but we can still make a small *correction* so that the cutter dominates one more agent. This is done by using a protocol, referred to as the CORRECTION protocol, which performs a careful redistribution. Finally, by running CORE one more time with a different cutter and the current residue, we show how further dominations arise that lead to the desired structure of one agent being dominated by two others. In total, phase one requires up to 6 calls to the CORE protocol.

Phase Two. Suppose that at the end of phase one, agent A is dominated by agents B and C. The goal in the second phase is to produce a partial envy-free allocation where both A and D dominate both B and C. To achieve this goal, we execute CORE twice on the residue with D as the cutter. Then, if we still do not have the required dominations, we use again the CORRECTION protocol to appropriately reallocate one of the last two partial allocations produced by CORE. This suffices to create the dominations shown in Fig. 1b.

Phase Three. Since both B and C are now dominated by A and D, we can simply execute the cut-and-choose protocol for B and C on the remaining residue.

Similarities and Differences with the Aziz-Mackenzie Algorithm [3]. Our algorithm uses similar building blocks as the algorithm in [3] for four agents, combined with new insights. Namely, our CORE and CORRECTION protocols on a high level serve the same purpose as the core and the permutation protocols in [3]. Conceptually, a crucial difference is the target structure of the domination graph. The initial (and most query-demanding) step of [3] is to have *every* agent dominate two other agents. Here, our goal in phase one is to *have just one agent dominated by two other agents*. Once this is accomplished, it is possible to reach a complete envy-free allocation much faster. Another important difference is the implementation of the CORE protocol itself. Our version is simpler regarding both its statement and its analysis. It also differs in the sense that it takes as input more information than in [3], such as the current allocation, and it is not required to always output a partial envy-free allocation of the current residue.

Fig. 1. Here is an illustration of the domination graphs we want to achieve at the end of the first (a) and the second (b) phase respectively. In both graphs *additional edges may be present* but are not relevant.

This extra flexibility allows us to avoid the tedious case analysis stated in the core protocol of [3] and, at the same time, further reduce the number of queries.

2 Preliminaries

Let $N = \{1, 2, 3, 4\}$ be a set of four agents. The cake is represented as the interval $[0, 1]$; a *piece* of the cake can be any finite union of disjoint intervals. Each agent $i \in N$ is associated with a valuation function v_i defined on all finite unions of intervals. We assume that the valuation functions satisfy the following standard properties for all $i \in N$:

- Normalization: $v_i([0, 1]) = 1$.
- Additivity: for all disjoint $X, X' \subseteq [0, 1]$: $v_i(X \cup X') = v_i(X) + v_i(X')$.
- Divisibility: for every $[x, y] \subseteq [0, 1]$ and every $\lambda \in [0, 1]$, there exists $z \in [x, y]$ such that $v_i([x, z]) = \lambda v_i([x, y])$. Note that this implies that $v_i([x, x]) = 0$, for all $x \in [0, 1]$.
- Nonnegativity: for every $X \subseteq [0, 1]$ it holds that $v_i(X) \geq 0$.

By $\mathcal{X} = (X_1, X_2, X_3, X_4)$ we denote the allocation where agent i is given the piece X_i.

Definition 1 (Envy-freeness). *An allocation $\mathcal{X} = (X_1, X_2, X_3, X_4)$ is* envy-free*, if $v_i(X_i) \geq v_i(X_j)$, for all $i, j \in N$, i.e., every agent prefers her piece to any other agent's piece.*

We say that $\mathcal{X}$ is a *partial* allocation, if there is some cake that has not been allocated yet, i.e., $\bigcup_{i=1}^{4} X_i \subsetneq [0, 1]$. The unallocated cake is called the *residue*. During the execution of the algorithm the residue diminishes, until eventually it becomes the empty set. As we noted, an important notion is that of *domination* or *irrevocable advantage* [6]. It will be insightful to think of a graph-theoretic representation of our goals, via the *domination graph* of the current allocation.

Definition 2 (Domination and Domination Graph). *Given a partial allocation $\mathcal{X} = (X_1, X_2, X_3, X_4)$ and a residue R, we say that an agent i* dominates *another agent j, if $v_i(X_i) \geq v_i(X_j \cup R)$. That is, i would not be envious of j even*

if j were allocated all of R. The domination graph *with respect to $\mathcal{X}$ is a directed graph where the nodes correspond to the agents and there exists a directed edge (i, j) if and only if agent i dominates agent j.*

Achieving certain patterns in the domination graph can make the allocation of the remaining residue straightforward. For example, if there exists a node i with in-degree 3, allocating all of the residue to agent i results in an envy-free allocation. As another example, the protocol of [3] tries to get a domination graph where every node has out-degree at least 2. In our algorithm, we also enforce a certain structure on the domination graph.

The Robertson-Webb Model. The standard model in which we measure the complexity of cake cutting algorithms is the one suggested by Robertson and Webb [14] and formalized by Woeginger and Sgall [19]. In this model, two kinds of queries are allowed:

- *Cut queries:* given an agent i, a point $x \in [0, 1]$ and a value r, with $r \leq v_i([x, 1])$, the query returns the smallest $y \in [0, 1]$ such that $v_i([x, y]) = r$.
- *Evaluation queries:* given an agent i and an interval $[x, y]$, return $v_i([x, y])$.

Virtually all known discrete protocols can be analyzed within this framework. For example, the cut-and-choose protocol is implemented by making one cut query for agent 1 with $r = 1/2$, starting from $x = 0$, followed by an evaluation query on agent 2 for one of the pieces (which also reveals the value of the second piece).

Conventions on Ties, Marks, Partial Pieces, and Residues. All algorithms in this work ignore ties. However, assuming an appropriate tie-breaking scheme, this is without loss of generality (also see the discussion in [2]).

We follow some conventions—also adopted in related work—when it comes to handling trims and partial pieces. In various steps during the algorithm, one agent cuts the residue into pieces, and the other agents are asked to place marks on certain pieces. We always assume that marks are placed starting from the left endpoint of a piece, and this operation creates a partial piece, contained between the mark and the right endpoint. In particular, suppose we have a partition of the residue into four contiguous pieces. Then, an agent may be asked to place a mark on her most favorite piece so that the resulting partial piece has the same value as her second favorite piece (see Fig. 2). The types of marks that the algorithm needs are described in the following definition.

Definition 3. *Given a partition of the residue into four pieces, we say that an agent performs an x-mark, if she places a mark on each of her $x - 1$ most valuable pieces so that the resulting partial pieces all have the same value as her x-th favorite piece.*

In the description of the algorithm we use 2-marks and 3-marks. Of course, after all marks are placed, each connected piece may have multiple marks on it.

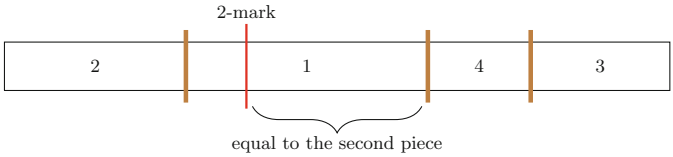

Fig. 2. The view of the residue for a non-cutter at the time she performs a 2-mark.

Whenever a connected piece p is only partially allocated, the part p' of p that is allocated is always the interval between the second rightmost mark and the right endpoint of p. While at this point it is not clear whether a *second* mark on a piece even exists, we show (see full version [1]) that marked pieces will have at least two marks. Hence, if some agent i receives a partial piece p', resulting from an initial piece p, it is not necessarily true that p' is defined by i's own mark.

Note that in the above discussion the residue is seen as a single interval, while in fact it may be a finite union of intervals. We keep this view throughout this work as it is conceptually easier and allows for a cleaner presentation.[1]

3 The Algorithm

Our main result is the following theorem. All missing proofs can be found in the full version of this work in [1].

Theorem 1. *The* Main Protocol *returns an envy-free allocation and makes at most* 61 *cut queries, and* 110 *evaluation queries.*

We discuss first the main steps of our algorithm and provide the relevant definitions and key properties.

Phase One. This is the most important part of the protocol, and computationally the most demanding one. The goal in phase one is to get a partial envy-free allocation, where some agent is dominated by two other agents, i.e., the underlying domination graph has a node with in-degree at least 2, as depicted in Fig. 1. In order to establish dominations among agents, we use a subroutine called Core protocol (stated in Sect. 3.1). This protocol takes as input a specified agent, called the *cutter*, the current partial allocation, and the current residue. The output of Core is a partial (*usually* envy-free) allocation of the residue with some additional properties described below. In the initial step of Core, the cutter divides the current residue into four equal-valued pieces according to her own valuation function. Throughout the protocol the rest of the agents—the *non-cutters*—may mark these pieces, and at the end, agents may be allocated either partial (marked) or complete pieces. Of course, if at any point Core outputs an envy-free allocation of the whole cake, the algorithm terminates.

[1] We elaborate further on this issue in our full version [1].

For technical convenience, CORE also takes as an input a subset of agents that we choose to exclude from *competition*. This roughly means that the excluded agents will choose their piece late in the CORE protocol because they dominate the other non-cutters. In most cases, this argument is just the empty set (and when no such argument is specified we mean that it is $\emptyset$). The full description of CORE is given in Sect. 3.1 and for now, we treat it as a black box and assume that it satisfies the following properties.

CORE**Property 1.** *The cutter and at least one more agent receive complete pieces, each worth exactly* $1/4$ *of the value of the current residue according to the cutter's valuation.*

CORE**Property 2.** *The allocation output by any single execution of* CORE *when no agent is excluded from competition, is a (possibly partial) envy-free allocation.*

The above properties allow us to deduce an important fact: if CORE is executed at least twice with the same agent as the cutter, then this cutter dominates at least one agent in the resulting allocation. In fact, we can be more specific about the agent who gets dominated. The important observation here is that a second run of CORE makes the cutter dominate whoever received the so-called *insignificant piece* in the first execution.

Definition 4. *Let* $\mathcal{A}$ *be an allocation produced by a single run of* CORE*. Among the four pieces given to the agents, the partial piece that is least desirable to the cutter is called the* insignificant piece *of* $\mathcal{A}$*.*

Hence, if we run CORE twice, say with agent 1 as the cutter, we enforce one edge in the domination graph. In order to proceed further and obtain a node with in-degree two, we first attempt, as an intermediate step, to have a domination graph where one node has out-degree equal to two. One may think that the intermediate step can be achieved by running CORE more times with agent 1 as the cutter. The problem with this approach is that even if we further execute CORE *any* number of times, there is no guarantee that new dominations will appear; the same agent may receive the insignificant piece in every iteration.

To fix this issue, it suffices to run CORE 4 times with agent 1 as the cutter and then make a small *correction* to one of the 4 partial allocations produced by CORE. In particular, denote by $\mathcal{A}^k = \{p_1^k, p_2^k, p_3^k, p_4^k\}$, with $k = 1, ..., 4$, the suballocation output by the kth execution of CORE within the *for* loop of line 1 of MAIN PROTOCOL, and let R^k be the residue after the kth execution. Then clearly for each agent i, $p_i^k \subseteq R^{k-1}$, and the current allocation of the algorithm after the 4 calls to CORE is $\mathcal{X} = \{p_1, p_2, p_3, p_4\}$, with $p_i = \cup_{k=1}^4 p_i^k$. Among these 4 suballocations that $\mathcal{X}$ consists of, we identify one in which we can perform a certain redistribution without introducing any envy. To do this, we exploit the notion of *gain*, which is the difference between the value that an agent has for her own piece compared to the pieces of agents she does not dominate.

Definition 5 (Gain). *Let* $\mathcal{X} = \{p_1, \ldots, p_4\}$ *be the current partial allocation of the cake, and* $\mathcal{A} = \{p'_1, \ldots, p'_4\}$ *be a suballocation of* $\mathcal{X}$*, i.e.,* $p'_i \subseteq p_i$ *for* $i \in N$*. Further, let* D_i *be the set of agents that are dominated by* i *in* $\mathcal{X}$ *and*

$N_i = N \setminus (D_i \cup \{i\})$. *Then the gain of* i *with respect to* $\mathcal{A}$, $\mathrm{G}_{\mathcal{A}}(i)$, *is the difference between* $v_i(p'_i)$ *and the maximum value of* i *for a piece in* $\mathcal{A}$ *given to any agent in* N_i, *i.e.,* $\mathrm{G}_{\mathcal{A}}(i) = v_i(p'_i) - \max_{j \in N_i} v_i(p'_j)$.[2]

Using Definition 5, we identify a suballocation among $\mathcal{A}^1, \mathcal{A}^2, \mathcal{A}^3, \mathcal{A}^4$, where the gain of each agent is small compared to her combined gain from the other three suballocations (line 4 of the algorithm). The existence of such an allocation is shown in the full version. Then, the redistribution is performed via the Correction protocol which takes as input an allocation $\mathcal{A}$, produced by Core, and outputs an allocation $\mathcal{A}' = \pi(\mathcal{A})$, where π is a permutation on N. In doing so, special attention is paid to the insignificant piece of $\mathcal{A}$. For now, we treat Correction as a black box and ask that it satisfies the three properties below; see Sect. 3.1 for its description.

Main Protocol(N)

Phase One

1 **for** *count* $= 1$ *to* 4 **do**
2 Run Core on the current residue with agent 1 as the cutter.
3 **if** *the same agent got the insignificant piece in all 4 executions of* Core **then**
4 Find $\mathcal{A}^* \in \{\mathcal{A}^1, \mathcal{A}^2, \mathcal{A}^3, \mathcal{A}^4\}$ such that $\mathrm{G}_{\mathcal{A}^*}(i) \leq \sum_{\mathcal{A} \neq \mathcal{A}^*} \mathrm{G}_{\mathcal{A}}(i)$ for all $i \in N \setminus \{1\}$.
5 Run Correction on $\mathcal{A}^*$.
6 Run Core on the residue with agent 1 as the cutter.
7 **if** *there is some agent* $E \in N \setminus \{1\}$ *not dominated by agent* 1 **then**
8 Run Core on the residue with agent E as the cutter, excluding agent 1 from competition (since agent 1 dominates the other non-cutters).
9 **else**
10 Run the Selfridge-Conway Protocol on the residue for agents 2, 3, and 4, and terminate.

Now, if the algorithm has not terminated, some agent A *is dominated by two other agents* B *and* C. *Let* D *be the remaining agent.*

Phase Two

11 **for** *count* $= 1$ *to* 2 **do**
12 Run Core on the current residue with agent D as the cutter, excluding from competition any one from $\{B, C\}$ who dominates two non-cutters.
13 **if** B *and* C *are not both dominated by* A *and* D **then**
14 Let $F \in \{B, C\}$ be the agent who got the insignificant piece in the last two calls of Core.
15 Run Correction on the suballocation (out of the last two) where $\mathrm{G}_{\mathcal{A}}(F)$ was smaller.

At this point both A *and* D *dominate both* B *and* C.

Phase Three

16 Run Cut and Choose on the current residue for agents B and C.

[2] Note that $\mathrm{G}_{\mathcal{A}}(i)$ is not defined when $N_i = \emptyset$. In fact, we never need it in such a case.

CorrectionProperty 1 *The insignificant piece of $\mathcal{A}$ is given to a different agent in $\mathcal{A}'$. In particular, it is given to an agent that has marked it in $\mathcal{A}$.*

CorrectionProperty 2 *If a non-cutter was allocated her favorite unmarked piece in $\mathcal{A}$, she will again be allocated a piece of the same value in $\mathcal{A}'$.*

Assume there is no agent dominating everyone else, meaning that $\mathrm{G}_{\mathcal{A}}(i)$ is defined for all $i \in N$. For a partial envy-free allocation like $\mathcal{A}$, the gain of any agent is nonnegative. However, this may not be true for $\mathcal{A}'$, as it is not necessarily envy-free. What we need is for $(\mathcal{X} \setminus \mathcal{A}) \cup \mathcal{A}'$ to be envy-free, and towards this $\mathrm{G}_{\mathcal{A}'}(i)$ should not be too small for any $i \in N$.

CorrectionProperty 3 $\mathrm{G}_{\mathcal{A}'}(i) \geq -\mathrm{G}_{\mathcal{A}}(i)$ for all agents i.

By Correction Property 1, the insignificant piece has changed hands after line 5. This allows us to make one extra call to Core in order to enforce one more domination (line 6). Hence, the intermediate step is completed and we know that agent 1 dominates at least 2 other agents. If she dominates all three of them, we can run any of the known procedures for 3 agents on the residue, and be done with only a few queries. The interesting remaining case is to assume that agent 1 currently dominates exactly two other agents.

At this point there are various ways to proceed. E.g., we could repeat the whole process so far, but with agents 2 and 3 as cutters, and get at least 6 edges in the domination graph. This would ensure a node with in-degree two, but it requires several calls to Core. Instead, and quite remarkably, we show that it suffices to run Core only one more time, with the agent who is not dominated by agent 1 as the cutter. In the full version of this paper we prove that this makes the cutter dominate one of the agents that are dominated by agent 1. Hence, phase one is now complete, as we have one agent with in-degree two.

Remark 1. The intermediate step of getting a node with out-degree two has also been utilized in [3]. The goal there however was to make *every* agent dominate two other agents, whereas we only needed this to hold for one agent.

Phase Two. Suppose phase two starts with a partial envy-free allocation where some agent, say A, is dominated by agents B and C (Fig. 1a). Our next goal is to produce a partial envy-free allocation where both A and D dominate both B and C (Fig. 1b). To achieve this goal, we execute Core twice with D as the cutter, i.e., with the agent not involved in the dominations of phase one. Again, we need to argue about the behavior of Core under the existing dominations, and we ask for the following property.

CORE**Property 3** *Assume we run* CORE *with* D *as the cutter, and suppose agent* A *is dominated by the other two non-cutters,* B *and* C*, neither of whom dominates the other. Then, (1)* A *gets her favorite of the four complete pieces without making any marks, (2) at least three complete pieces are allocated, and (3) if a non-cutter, say* B*, gets a partial piece, then the remaining non-cutter,* C*, is indifferent between her piece and* B*'s piece.*

Using this property, we can show that after one call to CORE (1st execution of line 12), agents A and D will both dominate either B *or* C. However, we need domination over both B *and* C. The second call to CORE (2nd execution of line 12) ensures that we can again resort to the CORRECTION protocol. If, after the two calls to CORE, only one of B and C, say B, is dominated by both A and D, then running CORRECTION on one of the two core allocations from this phase—the one where the gain of B is smaller—resolves the issue, and makes A and D dominate both B and C.

Phase Three. Since both agents B and C are dominated by A and D, we just execute the cut-and-choose protocol for B and C, where B cuts the residue in two equal pieces and C chooses her favorite piece. This completes our algorithm.

3.1 CORE and CORRECTION

Here we provide the description of the CORE and CORRECTION protocols. The main results within this subsection are the following.

Theorem 2. *The* CORE *protocol satisfies* CORE *Properties 1, 2 and 3, and makes at most* 9 *cut queries and* 15 *evaluation queries.*

Theorem 3. *The* CORRECTION *protocol satisfies* CORRECTION *Properties 1, 2 and 3, and makes no queries.*

CORE $(k, R, \mathcal{X}, \mathcal{E})$

Agent k cuts the current residue R in four equal-valued pieces (according to her).
Let $S = N \setminus (\{k\} \cup \mathcal{E})$ be the set of agents who may compete for pieces.
if *there exists $j \in S$ who has no competition in S for her favorite piece* **then**
    j is allocated her favorite piece and is removed from S.
if *every agent in S has a different favorite piece* **then**
    Everyone gets her favorite piece and the algorithm terminates.
for *every agent $i \in S$* **do**
    if *(1) i has no competition for her second favorite piece p,* **or**
    (2) i has exactly one competitor $j \in S$ for p, j also considers p as her second favorite, and i,j each have exactly one competitor for their favorite piece **then**
        i makes a 2-mark.
    else
        i makes a 3-mark.
Allocate the pieces according to a rightmost rule:
if *an agent has the rightmost mark in two pieces* **then**
    Out of the two partial pieces, considered until the second rightmost mark (which always exists), she is allocated the one she prefers.
    The other partial piece is given to the agent who made the second rightmost mark on it.
else
    Each partial piece is allocated—until the second rightmost mark—to the agent who made the rightmost mark on that piece.
if *any non-cutters were not given a piece yet* **then**
    Giving priority to any remaining agents in S (but in an otherwise arbitrary order), they choose their favorite unallocated complete piece.
The cutter is given the remaining unallocated complete piece.

The CORE protocol is used for allocating part of the current residue every time it is called. CORE takes as input an agent, specified as the cutter, the current residue, and the current partial allocation. It first asks the cutter to divide the residue into four equally valued contiguous pieces. The cutter is going to be the last one to receive one of these four pieces. Regarding the remaining three agents, each of them will either be immediately allocated her favorite piece or will be asked to place a mark on certain pieces, based on the relative rankings of the non-cutters for the pieces, and on possible domination relations that have already been established. Marks essentially provide limits on how to partially allocate pieces that are desired by many agents, so that they can be given without introducing envy. There are two possible types of marks that can be placed; 2-marks and 3-marks. The type of mark that the agents will be asked to place depends mainly on the conflicts that arise for the favorite and second favorite pieces of each agent.

Definition 6. *During an execution of* Core*, let P be a set of pieces and S be a subset of non-cutters. We say that an agent $i \in S$ has* competition *for a piece $p \in P$, if* (1) *i is not dominated by everyone in S, and* (2) *there exists $j \in S$ such that p is j's favorite or second favorite piece in P. We call j a* competitor *of i for p.*

Definition 6 helps us identify whether we need to perform a 2-mark or 3-mark on the available pieces. Furthermore, in some cases where we know that certain domination patterns appear, it is convenient to prevent some agents from competing for any piece (in particular, when some agent dominates the other non-cutters).

The Correction protocol takes as input an allocation $\mathcal{A}$, produced by Core, and outputs an allocation $\mathcal{A}' = \pi(\mathcal{A})$, where π is a permutation on N, so that the envy-freeness of the overall partial allocation and certain dominations are preserved. Its description is shown below and its analysis is provided in the full version. Note that we refer to the cutter in allocation $\mathcal{A}$ as D.

Correction($\mathcal{A}$)

Let A, B be the agents having the two marks on the insignificant piece, and suppose A was given this piece in allocation $\mathcal{A}$.
The insignificant piece is allocated to B.
if *there is no other partial piece* **then**
    Agents choose their favorite piece in the order C, A, D.
else
    Find the rightmost mark not made by B on the other partial piece. Let $E \in \{A, C\}$ be the agent who made it.
    Agent E is allocated the partial piece.
    The last non-cutter chooses her favorite among the two complete pieces.
    The cutter is allocated the remaining (complete) piece.

References

1. Amanatidis, G., Christodoulou, G., Fearnley, J., Markakis, E., Psomas, C.A., Vakaliou, E.: An improved envy-free cake cutting protocol for four agents (2018). https://arxiv.org/pdf/1807.00317.pdf
2. Aziz, H., Mackenzie, S.: A discrete and bounded envy-free cake cutting protocol for any number of agents. In: 57th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2016, pp. 416–427 (2016)
3. Aziz, H., Mackenzie, S.: A discrete and bounded envy-free cake cutting protocol for four agents. In: 48th ACM Symposium on the Theory of Computing, STOC 2016, pp. 454–464 (2016)
4. Aziz, H., Ye, C.: Cake cutting algorithms for piecewise constant and piecewise uniform valuations. In: Liu, T.-Y., Qi, Q., Ye, Y. (eds.) WINE 2014. LNCS, vol. 8877, pp. 1–14. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13129-0_1
5. Brams, S.J., Taylor, A.D.: An envy-free cake division protocol. Am. Math. Monthly **102**(1), 9–18 (1995)

6. Brams, S.J., Taylor, A.D.: Fair Division: from Cake Cutting to Dispute Resolution. Cambridge University Press, Cambridge (1996)
7. Branzei, S.: A note on envy-free cake cutting with polynomial valuations. Inf. Process. Lett. **115**(2), 93–95 (2015)
8. Dubins, L., Spanier, E.: How to cut a cake fairly. Am. Math. Monthly **68**, 1–17 (1961)
9. Kurokawa, D., Lai, J.K., Procaccia, A.D.: How to cut a cake before the party ends. In: Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2013, pp. 555–561 (2013)
10. Pikhurko, O.: On envy-free cake division. Am. Math. Monthly **107**(8), 736–738 (2000)
11. Procaccia, A.D.: Thou shalt covet thy neighbor's cake. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, pp. 239–244 (2009)
12. Procaccia, A.D.: Cake cutting algorithms. In: Brandt, F., Conitzer, V., Endriss, U., Lang, J., Procaccia, A.D. (eds.) Handbook of Computational Social Choice, chap. 13. Cambridge University Press (2016)
13. Robertson, J.M., Webb, W.A.: Near exact and envy-free cake division. Ars Combinatorica **45**, 97–108 (1997)
14. Robertson, J.M., Webb, W.A.: Cake Cutting Algorithms: be fair if you can. AK Peters (1998)
15. Segal-Halevi, E., Hassidim, A., Aumann, Y.: Waste makes haste: Bounded time algorithms for envy-free cake cutting with free disposal (2018). https://arxiv.org/pdf/1511.02599.pdf
16. Stromquist, W.: How to cut a cake fairly. Am. Math. Monthly **87**(8), 640–644 (1980)
17. Stromquist, W.: Envy-free cake divisions cannot be found by finite protocols. Electr. J. Comb. **15**(1) (2008)
18. Su, F.E.: Rental harmony: sperner's lemma in fair division. Am. Math. Monthly **106**(10), 930–942 (1999)
19. Woeginger, G., Sgall, J.: On the complexity of cake cutting. Discrete Optim. **4**(2), 213–220 (2007)