A Discrete and Bounded Locally Envy-Free Cake Cutting Protocol on Trees

Ganesh Ghalme, Xin Huang, Yuka Machino and Nidhi Rathi

October 6, 2023

A Discrete and Bounded Locally Envy-Free Cake Cutting Protocol on Trees

Ganesh Ghalme[1], Xin Huang[2], Yuka Machino[3], and Nidhi Rathi[4⋆]

[1] Indian Institute of Technology Hyderabad, India.
ganeshghalme@ai.iith.ac.in
[2] Technion Israel Institute of Technology, Haifa, Israel.
xinhuang@campus.technion.ac.il
[3] Massachusetts Institute of Technology, USA.
yukam997@mit.edu
[4] Max Planck Institute for Informatics, University of Saarland, Germany.
nrathi@mpi-inf.mpg.de

Abstract. We study the classic problem of *fairly* allocating a divisible resource modeled as a unit interval $[0, 1]$ and referred to as a *cake*. In a landmark result, Aziz and Mackenzie [4] gave the first discrete and bounded protocol for computing an *envy-free cake division*, but with a huge query complexity consisting of six towers of exponent in the number of agents, n. However, the best-known lower bound for the same is $\Omega(n^2)$, leaving a massive gap in our understanding of the complexity of the problem.

In this work, we study an important variant of the problem where agents are embedded on a graph whose edges determine agent relations. Given a graph, the goal is to find a *locally envy-free* allocation where every agent values her share of the cake at least as much as that of any of her *neighbors'* share. We identify a *non-trivial* graph structure, namely a tree having depth at most 2 (Depth2Tree), that admits a query efficient protocol to find locally envy-free allocations using $O(n^4 \log n)$ queries under the standard Robertson-Webb (RW) query model. To the best of our knowledge, this is the first such non-trivial graph structure. In our second result, we develop a novel cake-division protocol that finds a locally envy-free allocation among n agents on *any* Tree graph using $O(n^{2n})$ RW queries. Though exponential, our protocol for Tree graphs achieves a significant improvement over the best-known query complexity of six-towers-of-n for complete graphs.

1 Introduction

The problem of fairly dividing a set of resources among a set of participating agents is one of the most fundamental problems in distributive justice, with roots dating back to biblical time. However, arguably the first formal mathematical approach towards this problem was initiated by Steinhaus [24] (see also [14]). Over

⋆ A part of this work was done when N. Rathi was at Aarhus University, Denmark time, this problem has not only found interest in the academic communities of various disciplines such as social sciences, economics, mathematics, and computer science (see [9,10,23] for excellent expositions) but has also found relevance in a wide-range of real-world applications [12,16,21]. The problem of *cake division* provides an elegant abstraction to several situations where a divisible resource is to be allocated among agents with heterogeneous preferences such as division of land, allocation of radio and television spectrum, rent division, to name a few (see [19] for implementations of cake-cutting methods).

Formally, a cake-division instance consists of n agents, each having a cardinal preference over the cake that is modeled as a unit interval $[0, 1]$. These preferences are specified by a valuation function v_i's, and we write $v_i(I)$ to denote agent i's value for the piece $I \subseteq [0, 1]$. The goal is to partition the cake into n bundles (may consist of finitely many intervals) and assign them to the n agents such that this assignment is consider *fair*. A central notion of fairness in resource-allocation settings is *envy-freeness* that deems an allocation *fair* if every agent prefers her allocated share over that of any other agent [17]. The appeal of envy-freeness can be rightfully perceived from its strong existential result: under very mild assumptions, an envy-free cake division is guaranteed to always exist [15,25].

The algorithmic results for the problem, however, remain elusive. In fact, Stromquist [26] proved that there does not exist a finite protocol[5] for computing an envy-free cake division (with contiguous pieces) in an adversarial model. Furthermore, Deng et al. [13] showed that this problem is PPAD-hard when agents have ordinal valuations. The best-known protocol of Aziz and Mackenzie [4] for finding envy-free cake divisions with non-contiguous pieces has a super-exponential query complexity bound of $O(n^{n^{n^{n^{n^{n}}}}})$. In contrast, the best known lower bound for the problem is $\Omega(n^2)$ leaving a massive gap in our understanding of the query complexity of the problem [22] .

Current techniques do not seem well suited to yield an immediate answer to developing better protocols for finding envy-free cake divisions. Hence, one of the promising approach is to find efficient protocols for instances satisfying certain properties. In this work, we study one such interesting variant where we assume that envy comparisons between agents are dictated by an underlying graph over agents. The goal here is to find a *locally envy-free* allocation of the cake such that no agent envies her neighbor(s) in the given graph G. Note that when G is a complete graph, we retrieve the classical setting of envy-free cake division.

The above-described graphical framework opens various interesting directions for understanding the problem of fairness in cake division (see [1,7,8,11,27]). From a practical point of view, it captures many situations in which global knowledge is unavailable or unrealistic. For instance, when the graph represents social connections between a group of people, it is reasonable to assume that agents only envy the agents whom they know (i.e., friends or friends of friends). Similarly, when a graph represents rank hierarchy in an organization, it is reasonable to assume that agents only envy their immediate neighbors (i.e. colleagues).

[5] We consider the protocol under the standard Robertson-Webb query model.

The primary objective of this paper, however, is to study *local envy-freeness* from a theoretical standpoint. Given that the state-of-the-art protocol for finding envy-free allocation for a complete graph requires super-exponential queries, a natural line of research—and the focus of this work—is to identify interesting graph structures that admit faster protocols for computing locally envy-free allocations.

Our Results and Techniques: Our work focuses on cake-division instances where the underlying graph over the agents is either a Tree or a Depth2Tree. A Tree graph represents a setting where agents are embedded on a rooted tree (see Figure 1), while a Depth2Tree graph is a special case of tree graph with a depth at most two. Our protocols operate under the standard Robertson-Webb query model [23] (defined in Section 2), and hence are discrete in nature.

We begin in Section 3 by addressing an open problem listed in [1] and [7] that asks for identifying interesting classes of graph structures that admit polynomial-query algorithms for local envy-freeness. Our result identifies the first *non-trivial* graph structure over n agents, namely a Depth2Tree, for which we develop an efficient protocol for computing locally envy-free allocations. As a warm-up example, a simpler protocol for 4-agents-on-a-line graph and 5-agents-on-a-line graph can be found in the full version of the paper [18].

Theorem 1. *For cake-division instances with n agents on a* Depth2Tree*, there exists a discrete protocol that finds a locally envy-free allocation using at most $O(n^3 \log(n))$ `cut` and $O(n^4 \log(n))$ `eval` queries under RW model.*

Interestingly, the techniques developed for Depth2Tree graph do not extend trivially to the Tree graphs. Nonetheless, inspired by its main ideas, in Section 4, we develop a recursive protocol that computes a locally envy-free allocation among n agents on *any* Tree graph with a query complexity of $O(n^{2n})$. The idea of recursion imparts notable simplicity to our protocol, even though the analysis is somewhat intricate. Our next result addresses the open problem of designing a discrete and bounded protocol for local envy-freeness on trees mentioned in [1].

Theorem 2. *For cake-division instances with n agents on a* Tree*, there exists a discrete protocol that computes a locally envy-free allocation using at most $O(n^{2n})$ Robertson-Webb queries.*

1.1 Related Literature

The celebrated Selfridge-Conway protocol [23] finds an envy-free allocation among three agents using 5 queries. Aziz and Mackenzie [5] proposed a cake-cutting protocol that finds an envy-free allocation among four agents in (close to) 600 queries. This bound was then improved by Amanatidis et al. [2] to 171 queries. As discussed above, despite significant efforts, the problem of developing efficient envy-free cake-cutting (discrete) protocols for $n \geq 5$ agents remains largely open. Additionally, multiple hardness results have motivated various interesting settings. For example, efficient fair cake-cutting protocols for interesting classes of valuations have been developed in [6,20]. The work of Arunachaleswaran et al.

[3] developed an approximation algorithm that efficiently finds a cake division with contiguous pieces wherein the envy is multiplicatively bounded within a factor of $2 + O(1/n)$.

The problem of fair cake division with graphical (envy) constraints was first introduced by Abebe et al. [1]. They characterize the set of directed graphs for which an *oblivious* single-cutter protocol—a protocol that uses a single agent to cut the cake into pieces—admits a bounded query complexity for locally envy-free allocations in the Robertson-Webb query model. In contrast, our work studies a class of *undirected* graphs that are significantly harder to analyze, and surprisingly develop comparable upper bounds. In another closely related paper, Bei et al. [7] develops *a moving-knife protocol*[6] that outputs an envy-free allocation on tree graphs. In more recent work, Bei et al. [8] develop a discrete and bounded *locally proportional* protocol for any given graph. Tucker [27] compliments this result by providing a lower bound of $\Omega(n^2)$ on the query complexity of obtaining locally proportional allocation in the Robertson-Webb model. In contrast, our work addresses a stronger guarantee of local envy-freeness. We address the open questions raised in [1,8] by (a) developing a discrete and bounded protocol for Tree graphs with single-exponential query complexity, and (b) constructing a query-efficient discrete protocol that finds locally envy-free allocations among n agents on Depth2Tree.

2 The Setting

We write $[k]$ to denote the set $\{1, 2, \ldots, k\}$ for a positive integer k.

This work considers the problem of fairly allocating a divisible resource—modeled as a unit interval $[0, 1]$ and referred to as a *cake*—among n agents denoted by $(a_1, a_2, \ldots, a_n)$. For an agent a_i for $i \in [n]$, we write v_i to specify her cardinal valuations over the intervals in $[0, 1]$. In particular, $v_i(I) \in \mathbb{R}^+ \cup \{0\}$ represents the valuation of agent a_i for the interval $I \subseteq [0, 1]$. For brevity, we will write $v_i(x, y)$ instead of $v_i([x, y])$ to denote agent a_i's value for an interval $[x, y] \subseteq [0, 1]$. Following the standard convention, we assume that v_is are non-negative, additive,[7] and non-atomic.[8] Additionally, without loss of generality, we assume that the valuations are normalized i.e., $v_i(0, 1) = 1$ for all $i \in [n]$.

We write G to denote the underlying graph structure over the agents, often written as *agents are on G*. Here, the nodes of G represent agents and the edges among them correspond to *envy constraints*. We will write $\mathcal{I}$ to refer to a cake-division instance with graph constraints.

2.1 Preliminaries

For cake-division instance with n agents we define an *allocation* $\mathcal{A} = (A_1, A_2, \ldots, A_n)$ of the cake $[0, 1]$ to be a collection of n pair-wise disjoint pieces such that

[6] A moving-knife protocol may not be implementable in discrete steps under standard RW query model, and hence our result for Tree graphs is stronger than that of [7].

[7] For any two disjoint intervals $I_1, I_2 \subseteq [0, 1]$, we have $v_i(I_1 \cup I_2) = v_i(I_1) + v_i(I_2)$.

[8] For any interval $[x, y]$ and any $\lambda \in [0, 1]$, $\exists y' \in [x, y]$ such that $v_i(x, y') = \lambda \cdot v_i(x, y)$.

$\cup_{i\in[n]} A_i = [0, 1]$. Here, a piece or a bundle A_i (a finite union of intervals of the cake $[0, 1]$) is assigned to agent a_i for $i \in [n]$. We say $\mathcal{A}$ is a *partial* allocation if the union of A_is forms a strict subset of $[0, 1]$. In this work, we study the fairness notion of *local envy-freeness* defined below.

Definition 1 (Local Envy-freeness). *Given a cake-division instance with a graph G, an allocation $\mathcal{A} = (A_1, A_2, \cdots, A_n)$ is said to be* locally envy-free *(on G) if for any agent a_i for $i \in [n]$ and any piece $A_j \in \mathcal{A}$ such that a_i and a_j have an edge in G, we have $v_i(A_i) \geq v_i(A_j)$. When G is a complete graph, a locally envy-free allocation is called as an* envy-free allocation*.*

Definition 2 (Robertson-Webb (RW) query model). *Our protocols operate under the standard Robertson-Webb query model [23] that allows access agents' valuations via the following two types of queries:*[9]

1. *Cut query: Given a point $x \in [0, 1]$ and a target value $\tau \in [0, 1]$, $\texttt{cut}_i(x, \tau)$ asks agent a_i to report the smallest $y \in [x, 1]$ such that $v_i(x, y) = \tau$. If such a y does not exist, then the response is some arbitrary number, say -1.*
2. *Evaluation query: Given $0 \leq x < y \leq 1$, $\texttt{eval}_i(x, y)$ asks agent a_i to report her value $v_i(x, y)$ for the interval $[x, y]$ of the cake.*

Our protocols use the following three standard procedures (formal descriptions are deferred to the full version of the paper [18]).
Select: Given a collection of pieces $\mathcal{X}$, $m \leq |\mathcal{X}|$, and an agent a_i, SELECT($\mathcal{X}, a_i, m$) returns the m highest-valued pieces in $\mathcal{X}$ according to v_i. It is easy to see that SELECT requires zero `cut` queries and at most $|\mathcal{X}|$ `eval` queries.
Trim: Given a collection of pieces $\mathcal{X}$ and an agent a_i, TRIM($\mathcal{X}, a_i$) returns a collection of $|\mathcal{X}|$-many piece, each of value equal to her smallest-valued piece in $\mathcal{X}$ along with some *residue* R. It first finds the lowest valued piece according to v_i and makes the remaining pieces of value equal to it by trimming. The *residue* is the collection of all the trimmings formed in the procedure. The TRIM procedure requires $|\mathcal{X}| - 1$ `cut` queries and $|\mathcal{X}|$ `eval` queries.
Equal: Given a collection of pieces $\mathcal{X}$ and an agent a_i, EQUAL($\mathcal{X}, a_i$) redistributes among the pieces in $\mathcal{X}$ (creating no residue) such that each piece is equally valued by a_i. It also identifies a bundle in the original collection that has a value larger than the average value of the bundles (according to v_i). Note that while both EQUAL and TRIM procedures return an allocation where all the pieces are equally valued by a_i, TRIM may generate a residue whereas EQUAL redistributes the pieces of the cake without leaving any residue. The EQUAL procedure requires $|\mathcal{X}| - 1$ `cut` queries and $|\mathcal{X}|$ `eval` queries.

3 Depth2Tree: A Tractable Instance

In this section, we consider cake-division instances where the underlying graph over the agents is a DEPTH2TREE and develop a novel protocol to compute

[9] See a remark in the full version of the paper [18].

locally envy-free allocations among n agents using poynomially-many queries (Theorem 1). We assume that the graph is rooted at agent a_r and we write D to denote the set of neighbours/children of agent a_r. Each agent $a_i \in D$ has $\ell_i + 1$ neighbours, i.e., she is connected to $\ell_i \geq 0$ leaf agents. In addition, we write $L(i)$ to denote the set of children of agent $a_i \in D$.

Overview of the* D2Tree *protocol: For cake-division instances with n agents on a Depth2Tree, we develop a protocol D2Tree that progressively builds an allocation $\mathcal{A} = (A_1, \ldots, A_n)$ among n agents. D2Tree primarily consists of a while-loop that has three phases: `Selection`, `Trimming`, and `Equaling`. The protocol maintains a set $\mathrm{Tr} \subseteq D$ of agents who perform the Trim operation during the execution of our protocol.

We initialize the set $\mathrm{Tr} = D$ to contain all the neighbour agents of a_r. The key goal of the while-loop is to create *domination* (see Defn. 3) for agent a_r over her each neighbour in D one by one. An agent $a_i \in \mathrm{Tr}$ gets removed from this set as soon as a_r *dominates* her; we show that the set Tr shrinks as the algorithm progresses (Lemma 1). And finally, the while-loop terminates when $\mathrm{Tr} = \emptyset$.

We will call the unallocated part of the cake obtained at the end of each round of the while-loop as *residue*, denoted by R. In the beginning, the residue $R = [0, 1]$ and it keeps decreasing with subsequent rounds of the while-loop. Throughout the algorithm, each agent $a_i \in D$ maintains a set $\mathcal{A}^{(i)} = (A_0^{(i)}, \ldots, A_{\ell_i}^{(i)})$ of $\ell_i + 1$ bundles of equal value to her. Note that the leaf agents do not perform any operation throughout the entire execution of the while-loop.

Definition 3 (D2Tree Domination Condition). *At any round of the while-loop with residue R, we say that agent a_r with bundle A_r* dominates *her neighbour agent $a_i \in D$ with a collection $\mathcal{A}^{(i)} = (A_0^{(i)}, \ldots, A_{\ell_i}^{(i)})$ of $\ell_i + 1$ bundles if*

1. *For bundle $A_0^{(i)} \in \mathcal{A}^{(i)}$, we have $v_r(A_r) = v_r(A_0^{(i)})$*
2. *For all the remaining bundles in the set $\mathcal{A}^{(i)}$, we have $v_r(A_r) - v_r(A_k^{(i)}) \geq v_r(R)$ for all $k \in [\ell_i]$*

The domination condition says that bundle A_r has become sufficiently more valuable than the bundles $A_k^{(i)} \in \mathcal{A}^{(i)}$ for $k \in [\ell_i]$ according to agent a_r such that even after the whole of residue (of that round) is added to any bundle in $\mathcal{A}^{(i)}$, a_r will not envy the recipient of that bundle.

Any round t of the while-loop with residue $R = R^t$ and the current set Tr of the trimmer agents consists of the following three phases:

Selection Phase: In the beginning, agent a_r divides the residue R^t into n equal pieces, each of value $v_r(R^t)/n$ to her; we denote this set by $\mathcal{X}$ (in Step 3). Then, one by one, each agent $a_i \in D$ selects her $\ell_i + 1$ most favorite (available) pieces from $\mathcal{X}$ (see Step 5). We denote the set of these selected pieces by $\mathcal{X}^{(i)} = \{X_0^{(i)}, \ldots, X_{\ell_i}^{(i)}\} \subseteq \mathcal{X}$, where $X_0^{(i)}$ is a_i's least valued piece in $\mathcal{X}^{(i)}$. Note that, we have $v_r(\mathcal{X}^{(i)}) = (\ell_i + 1) \cdot v_r(R^t)/n$.

After every neighbour agent of a_r in D has made her selection, the remaining single piece (of value $v_r(R^t)/n$) from $\mathcal{X}$ is added to the bundle A_r (in Step 6).

Trimming Phase: This phase begins with every agent $a_i \in Tr$ adding her least-valued piece, $X_0^{(i)} \in \mathcal{X}^{(i)}$ to bundle $A_0^{(i)}$ (in Step 12). This implies that

$$v_r(A_r) = v_r(A_0^{(i)}) \tag{1}$$

and hence the first condition of domination will be satisfied. This is due to the fact that both the bundles A_r and $A_0^{(i)}$ get a piece of value $v_r(R^t)/n$ in each round t. For the remaining ℓ_i bundles, agent $a_i \in \mathrm{Tr}$ performs a Trim procedure, making these ℓ_i bundles of value equal to $v_i(X_0^{(i)})$, and forming the set $\mathcal{Y}^{(i)}$ (see Step 10). The residue due to this operation is added to the residue R^{t+1} for the next round $t+1$ (Step 11). All the bundles in $\mathcal{Y}^{(i)}$ obtained from the `Trimming` phase are added one to each bundle in $\mathcal{A}^{(i)}$ in a way that helps us achieve the desired dominance (see Steps 12-14). Due to the Trim operation, we have

$$v_r(A_r) \geq v_r(A_k^{(i)}) \text{ for all } k \in [\ell_i] \tag{2}$$

We show that after repeated applications of the Trim operation, agent a_r achieves domination over agent a_i. In particular, we prove (in Lemma 1) that after every $O(n \log n)$ rounds of the while-loop, there exists some agent $a_i \in \mathrm{Tr}$ over whom dominance is achieved.

Equaling Phase: Let agent a_r achieves dominance over some agent $a_i \in D$ in round $t-1$, after which she is removed from the set Tr. That is, we have $v_r(A_r) - v_r(A_k^{(i)}) \geq v_r(R^t)$ for all $k \in [\ell_i]$, where[10] R^t is the residue formed at the end of round $t-1$, and hence is the residue at the beginning of round t.

In round t, agent a_i still begins with the Selection phase as before. Her bundle $A_0^{(i)}$ receives a trimmed piece from her set $\mathcal{Y}^{(i)}$; see Steps 18-19. Therefore, for all the subsequent rounds, we obtain

$$v_r(A_r) \geq v_r(A_0^{(i)}) \tag{3}$$

She next performs the Equal operation (in Step 17) which does not produce any residue. We will show that due to the second condition of the dominance (Definition 3), no matter how the residue of future rounds is distributed among the bundles of $\mathcal{A}^{(i)}$, agent a_r will have no envy towards any of the bundles formed during this procedure.

Termination of the while-loop: Once the set Tr becomes empty and agent a_r dominates every agent $a_i \in D$, the while-loop terminates. The final allocation is formed in Steps 24-27: agent a_r receives bundle A_r, each leaf agent $a \in L_i$ corresponding to agent $a_i \in D$ chooses her favorite bundle from the set $\mathcal{A}^{(i)}$ formed by her parent agent a_i, while a_i receives the last remaining bundle in the above set. This creates a complete allocation $\mathcal{A}$ of the cake that we will show is locally envy-free on Depth2Tree.

[10] Here, the bundles A_r and $A_k^{(i)}$ for $k \in [\ell_i]$ are the ones that were formed till the end of round $t-1$. For brevity, we do not add the notation $t-1$ in these bundles.

D2Tree: Local Envy-freeness for n agents on DEPTH2TREE

Input: A cake-division instance $\mathcal{I}$ on DEPTH2TREE $=(n, a_r, D, \{\ell_i\}_{a_i \in D})$
Output: A locally envy-free allocation.

Initialize $R \leftarrow [0,1]$, set of trimmer agents $\text{Tr} \leftarrow D$, bundles $A_0^{(i)}, \ldots, A_{\ell_i}^{(i)} \leftarrow \emptyset$ for each $a_i \in D$ and a bundle $A_r \leftarrow \emptyset$ for the root agent.
while $\text{Tr} \neq \emptyset$ **do**
  Agent a_r divides R into n equally-valued pieces that are kept in the set $\mathcal{X}$
  ——— `Selection` ———
  for $a_i \in D$ **do**
    $\mathcal{X}^{(i)} \leftarrow \text{SELECT}(a_i, \mathcal{X}, \ell_i + 1)$
  Set $A_r \leftarrow A_r \cup (\mathcal{X} \setminus \cup_{a_i \in D}(\mathcal{X}^{(i)}))$ `/* The remaining single piece from` $\mathcal{X}$ `*/`
  ——— `Trimming` ———
  Set $R \leftarrow \emptyset$
  for $a_i \in \text{Tr}$ **do**
    Let $X_0^{(i)} = \arg\min_{X \in \mathcal{X}^{(i)}} v_i(X)$ `/* This piece won't be trimmed */`
    $(\mathcal{Y}^{(i)}, R') \leftarrow \text{TRIM}(a_i, \mathcal{X}^{(i)})$
    Set $R \leftarrow R \cup R'$
    $A_0^{(i)} \leftarrow A_0^{(i)} \cup X_0^{(i)}$ and $\mathcal{Y}^{(i)} \leftarrow \mathcal{Y}^{(i)} \setminus X_0^{(i)}$
    Let $A_w^{(i)} = \arg\max_{1 \leq k \leq \ell_i} v_r(A_k^{(i)})$ and $Y_g^{(i)} = \arg\min_{1 \leq k \leq \ell_i} v_r(Y_k^{(i)})$
    $A_w^{(i)} \leftarrow A_w^{(i)} \cup Y_g^{(i)}$ and $\mathcal{Y}^{(i)} \leftarrow \mathcal{Y}^{(i)} \setminus Y_g^{(i)}$ `/* Trying to achieve domination on` $A_w^{(i)}$ `for the root agent */`
    For each $k \neq 0, w$, add one arbitrary piece from $\mathcal{Y}^{(i)}$ to $A_k^{(i)}$
  ——— `Equaling` ———
  for $a_i \in D \setminus \text{Tr}$ **do**
    $(\mathcal{Y}^{(i)}, Y_*) \leftarrow \text{EQUAL}(a_i, \mathcal{X}^{(i)})$
    Let $Y_k^{(i)} \in \mathcal{Y}^{(i)}$ be the piece such that $Y_k^{(i)} \subseteq Y_*$ `/* There is only one piece satisfying this condition. */`
    $A_0^i \leftarrow A_0^i \cup Y_k^{(i)}$ `/* This ensures that root will not envy the bundle` A_0^i `*/`
    For each $k \neq 0$, add one arbitrary piece from $\mathcal{Y}^{(i)}$ to the bundle $A_k^{(i)}$
  ——— `Checking Domination` ———
  for $a_i \in \text{Tr}$ **do**
    if *agent a_r dominates agent a_i (see Definition 3)*
      $\text{Tr} \leftarrow \text{Tr} \setminus \{i\}$
——— `Choose after while loop` ———
for $a_i \in D$ **do**
  for $a_j \in L(i)$ **do**
    a_j is allocated her favorite (available) bundle from $(A_k^{(i)})_{k \in [\ell_i]}$
  a_i is allocated the remaining bundle
return The allocation $A_r \cup (A_k^{(i)})_{k \in [\ell_i], a_i \in D}$

Theorem 1. *For cake-division instances with n agents on a* DEPTH2TREE*, there exists a discrete protocol that finds a locally envy-free allocation using at most $O(n^3 \log(n))$* `cut` *and $O(n^4 \log(n))$* `eval` *queries under RW model.*

Proof We begin by establishing three important properties of D2Tree (in Lemma 1) that are crucial in establishing the desired polynomial upper bound on its query complexity. Their proofs appear in the full version of the paper [18].

Lemma 1. *The following properties hold true for* D2Tree *protocol:*

1. *In every round of the while-loop,* D2Tree *protocol makes $O(n)$ cuts on the cake using $O(n)$ `cut` and $O(n^2)$ `eval` queries.*
2. *Agent a_r's valuation for the residues in two consecutive rounds t and $t+1$ of the while-loop in* D2Tree *satisfies $v_r(R^{t+1}) \leq (1 - (|D|+1)/n)v_r(R^t)$.*
3. *Agent a_r starts dominating at least one agent from the set* Tr *in every $O(n \log n)$ rounds of the while-loop in* D2Tree*, after which it is removed from the set* Tr*. That is, the size of the set* $|\mathrm{Tr}|$ *decreases by one in every $O(n \log n)$ rounds of the while-loop.*

For a given cake-division instance, we will prove that the output allocation $\mathcal{A} = (A_1, \ldots, A_n)$ of D2Tree is locally envy-free by splitting the analysis into three following cases.

(a) **Root agent:** Consider an arbitrary agent $a_i \in D$ and the set $\mathcal{A}^{(i)}$ of ℓ_i many bundles formed after the termination of the while-loop in D2Tree. Note that, agent a_i is assigned one bundle from the set $\mathcal{A}^{(i)}$. Therefore, to prove local envy-freeness for agent a_r, it suffices to show that agent a_r prefers her own bundle A_r over any bundle in the set $\mathcal{A}^{(i)}$. Note that, equations (1) and (2) imply $v_r(A_r) \geq v_r(A_k^{(i)})$ for all $k \in \{0\} \cup [\ell_i]$ throughout the trimming phase of agent a_i. Once agent a_r dominates agent a_i, say with respect to residue R^t of round t, we must have $v_r(A_r) \geq v_r(A_k^{(i)}) + v_r(R^t)$ for all $k \in [\ell_i]$. Therefore, agent a_r will not envy any of the bundles in the set $\mathcal{A}^{(i)}$ in future rounds $t' \geq t+1$ irrespective of how residue $R^{t'}$ is divided into these bundles in the equaling phase of agent a_i. Furthermore, recall that equation (3) ensures that agent a_r does not envy the bundle $A_0^{(i)}$ as well. Overall, agent a_r has no local envy in the final allocation.

(b) **Neighbour agents:** Consider an arbitrary agent $a_i \in D$. Throughout the execution of our algorithm, every bundle $A_k^{(i)} \in \mathcal{A}^{(i)}$ is of equal value to agent a_i. This follows due to the properties of Trim and Equal operations. Therefore, when the leaf agents of agent a_i (in the set L_i) selects her bundle from $\mathcal{A}^{(i)}$ in Step 25, agent a_i will have no envy towards them.

Next, we will show that a_i has no envy towards the root agent as well. Recall that in the selection phase, a_i chooses her favourite $\ell_i + 1$ pieces from $\mathcal{X}$ (in Step 5). The remaining single piece is added to agent a_r's bundle A_r. Therefore, in each round of the while-loop, the increment for each bundle $A_k^{(i)}$ for $k \geq 0$ is as large as the increment in bundle A_r in the view of agent a_i. That is, we have $v_i(A_k^{(i)}) \geq v_i(A_r)$ for all $k \in \{0\} \cup [\ell_i]$, and hence no local envy for agent a_i.

(c) **Leaf agents:** It is trivial to observe that any leaf agent will have no local envy since she chooses her favourite bundle before her neighbour agent.

Overall, D2Tree outputs a locally envy-free allocation among n agents on a Depth2Tree.

Counting Queries: Lemma 1 ensures that after $O(n \log n)$ rounds, the number of agents $a_i \in D$ who are in the set Tr decreases at least by one. Hence, the while-loop terminates (i.e. when Tr $= \emptyset$) after at most $O(n^2 \log n)$ many rounds. By Lemma 1, we know that each round of the while-loop requires $O(n)$ `cut` and $O(n^2)$ `eval` queries. Hence, D2Tree requires $O(n^3 \log n)$ `cut` queries and $O(n^4 \log n)$ `eval` queries. This completes our proof. □

4 Local Envy-freeness on a Tree

In this section, we develop a recursive protocol Domination(R, k) that finds a locally envy-free allocation for n agents on a Tree graph using at most $O(n^{2n})$ RW queries (Theorem 2). Without loss of generality, we *always* assume that the agents $a_1, a_2, \ldots, a_n$ are indexed according to some arbitrary topological ordering; making agent a_n as the root node and a_1 a leaf node in the graph. The topological order over the agents ensures that any descendant of agent a_j for $j \in [n]$ must have an index smaller than j.
Terminology: For a given cake-division instance with n agents on a Tree, we define the following sets for any $j \in [n]$.

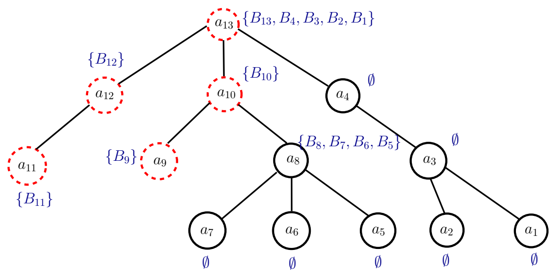
1. $D_j := \{j\} \cup \{i \in [j-1] : a_i \text{ is a descendant of } a_j\}$ is the set containing index j and the indices of descendants of a_j in the underlying Tree. We write $d_j = |D_j|$ to denote the size of the set D_j.
2. C_j denotes the set of indices of the immediate descendants (or children) of a_j in the underlying Tree. Furthermore, we write p_j to denote the index of the parent of agent a_j.

For a given fixed index $k \in [n]$, we say an agent a_j is *active* if $j \geq k+1$, otherwise she is *inactive*. Next we define important sets used in the analysis of the protocol.

1. For an active agent a_j, $\texttt{Inchild}(k, a_j) := \{i \mid i \leq k \text{ and } i \in C_j\}$ is the set consisting the indices of all her inactive children. This set is empty for inactive agents.
2. For an active agent a_j, $\texttt{Inact}(k, a_j) := \{j\} \cup \{t \in D_i \mid i \in \texttt{Inchild}(k, a_j)\}$ is the set consisting of the indices of all the descendants of her inactive children. This set is empty for inactive agents.
3. For an allocation $\mathcal{B} = (B_1, \ldots, B_n)$, we define $\texttt{Storage}(k, a_j) := \{B_i \mid i \in \texttt{Inact}(k, a_j)\}$ as the set that stores the bundles assigned to agents whose indices are in $\texttt{Inact}(k, a_j)$.

For an allocation $\mathcal{B}$, the collection of storage sets $\{\texttt{Storage}(k, a_j) \text{ for } j \in [n]\}$ creates a partition of its bundles. When the value of k is obvious from the context, we will use the phrase *storage of* a_j to refer to her $\texttt{Storage}(k, a_j)$ set.

Definition 4 (k-Fair allocation for Trees). *Consider a cake-division instance with n agents on a* Tree. *For a given piece $R \subseteq [0, 1]$, we say an allocation $\mathcal{B} = (B_1, \ldots, B_n)$ (of R) is k-*Fair *if for each agent a_j with $j \geq k$ the following conditions hold.*



	$\texttt{Inchild}(7, a_j)$	$\texttt{Inact}(7, a_j)$	$\texttt{Storage}(7, a_j)$
$j \leq 7$	$\emptyset$	$\emptyset$	$\emptyset$
$j = 8$	$\{5, 6, 7\}$	$\{5, 6, 7, 8\}$	$\{B_5, B_6, B_7, B_8\}$
$9 \leq j \leq 12$	$\emptyset$	$\{j\}$	$\{B_j\}$
$j = 13$	$\{4\}$	$\{1, 2, 3, 4, 13\}$	$\{B_1, B_2, B_3, B_4, B_{13}\}$

Fig. 1: The left figure depicts a representative example with 13 agents on a Tree. The agent corresponding to every node is written inside the circle. For a fixed index $k = 8$, the red dashed nodes and the black nodes represent the active and inactive agents respectively. The adjoining table details the sets $\texttt{Inchild}(k-1, a_j)$, $\texttt{Inact}(k-1, a_j)$, and $\texttt{Storage}(k-1, a_j)$ for an allocation $\mathcal{B} = (B_1, \ldots, B_n)$ with respect to $k = 8$. The set written next to a_j in the figure is her $\texttt{Storage}(k-1, a_j)$.

C1. Agent a_j does not envy her neighbours.
C2. $v_j(B_j) = v_j(B)$ for all $B \in \texttt{Storage}(k-1, a_j)$, and
C3. $v_j(B_j) \geq v_j(B)$ for all $B \in \texttt{Storage}(k-1, a_\ell)$ such that a_ℓ is an active child of a_j (with respect to index $k-1$).

The above-defined notion of k-Fairness forms the crux of our technical ideas. We explain this notion with the following example (Example 1).

Example 1. Consider an instance of a Tree graph with 13 agents depicted in Figure 1.

An 8-Fair allocation $\mathcal{B} = \{B_1, \ldots, B_{13}\}$ of some $R \subseteq [0, 1]$ satisfies the following conditions.

- *C1:* Agents $a_8, a_9, \ldots, a_{13}$ do not envy their neighbors.
- *C2:* We have $v_8(B_8) = v_8(B)$ for all $B \in \{B_5, B_6, B_7, B_8\}$ and $v_{13}(B_{13}) = v_{13}(B)$ for all $B \in \{B_1, B_2, B_3, B_4, B_{13}\}$.
- *C3:* We detail the condition for $j = 10$, and other cases can be dealt similarly. Agent a_{10} has two active children (with respect to index $k - 1 = 7$): a_9 and a_8 with storage sets $\{B_9\}$ and $\{B_5, B_6, B_7, B_8\}$ respectively. Hence, for $j = 10$ we have $v_{10}(B_{10}) \geq v_{10}(B)$ for $B \in \{B_5, B_6, B_7, B_8, B_9\}$.

Note that any 1-Fair allocation is locally envy-free for agents on a Tree. For any piece $R \subseteq [0, 1]$ and index $k \geq 1$, we will develop a recursive protocol Domination(R, k) that is always k-Fair (see Lemma 2). Hence, Domination$([0, 1], 1)$ will output the desired locally envy-free allocation among n agents on a Tree.

Overview of Domination(R, k)**:** For $k = n$ and any piece $R \subseteq [0, 1]$, protocol Domination(R, n) is defined in a straight-forward manner: agent a_n cuts R into n equal pieces according to her. It is easy to see that this allocation is indeed n-Fair. For $k \geq 1$, our protocol Domination(R, k) successively constructs a k-Fair allocation $\mathcal{A} = (A_1, \ldots, A_n)$ of R among n agents in multiple rounds. It does so by repeatedly invoking DOMINATION$(R, k+1)$ and using its $(k+1)$-FAIR output allocations. We will refer to R as *residue* and it keeps on shrinking as the algorithm proceeds. DOMINATION(R, k) primarily consists of a while-loop that terminates when the residue reduces so much that the parent agent a_{p_k} (of agent a_k) satisfies a certain *domination condition* (as stated in Step 8 of DOMINATION(R, k)) over the bundles of agents in D_k with respect to the current residue. This *domination* serves as a crucial property for creating the desired k-FAIR allocation (without creating any envy for agent a_{p_k}). We prove that the domination is achieved in polynomial many rounds of the while-loop (see Lemma 2) that becomes the backbone argument to establish the desired query complexity of our protocol DOMINATION$([0,1], 2)$.

Let us consider an arbitrary round t of the while-loop (Step 3-17) during the execution of DOMINATION(R, k), and denote the residue at its beginning as R^t; where $R^1 = R$. The round begins with invoking DOMINATION$(R^t, k+1)$ to obtain a $(k+1)$-FAIR allocation $\mathcal{B}^t = (B_1^t, B_2^t, \ldots, B_n^t)$ of R^t. Throughout round t, we focus (and modify *some* of) the bundles in the $\texttt{Storage}(k, a_{p_k})$ set corresponding to $\mathcal{B}^t$. Recall that, due to $(k+1)$-FAIRness of $\mathcal{B}^t$, agent a_{p_k} values each bundle in her $\texttt{Storage}(k, a_{p_k})$ set equally. Also, note that $\texttt{Storage}(k, a_k)$ corresponding to $\mathcal{B}^t$ is empty and that is exactly what agent a_k is trying to build during this recursive step. The challenge is to form the $\texttt{Storage}(k-1, a_k)$ set corresponding to the output allocation $\mathcal{A}$ while ensuring its k-FAIRness. Observe that, this allocation $\mathcal{A}$ (and its $\texttt{Storage}(k-1, a_k)$ set) is then used by DOMINATION$(R, k-1)$ in the next recursive step.

The while-loop of our protocol consists of three phases: `Selection`, `Trimming`, and `Equaling` where only the agent a_k performs the associated operations.

- In the first phase of SELECTION, as the name suggests, agent a_k *selects* her $|D_k| = d_k$-many favorite pieces from the $\texttt{Storage}(k, a_{p_k})$ set (see Step 4), denoted by the set $\mathcal{X}^t$.[11] Since agent a_{p_k} values each bundle in her storage set equally, we can re-index all the selected bundles in $\mathcal{X}^t$ to match the indices in the set D_k (and accordingly re-index the remaining non-selected bundles as well). Now, for every $j \notin D_k$, the *intact*[12] bundle B_j is added to the bundle A_j of agent a_j. We use this fact to establish Condition C1 of k-FAIRness for output allocation $\mathcal{A}$ (in Lemma 2).
- If agent a_{p_k} has not yet achieved the *domination* (as stated in Step 8), then agent a_k enters into the `Trimming` phase and performs a TRIM operation (Step 11) on the set $\mathcal{X}^t$ of bundles chosen in Step 5 to obtain a trimmed set $\mathcal{Y}^t$ of d_k-many bundles. The residue obtained due to this trimming process becomes the residue R^{t+1} for the next round. The bundles in $\mathcal{Y}^t$ are allocated (in Steps 11-14) among agents in D_k in a way that expedites the desired domination and ensures Condition C3 of k-FAIRness for allocation $\mathcal{A}$.

[11] Recall that the set D_k consists of the indices of the descendants of agent a_k and her own index. Since the indices of the agents follow topological ordering, $d_k \leq k$ for any $k \in [n]$. Moreover, since $p_k > k$, the set $\texttt{Storage}(k, a_{p_k})$ set cannot be empty.

[12] A bundle is said to be *intact* if it is in the form as present in $\mathcal{B}^t$ obtained from DOMINATION$(R^t, k+1)$ and has not been modified.

- The key observation here is that the value of the residue according to agent a_{p_k} decreases exponentially fast. This ensures that the said domination for agent a_{p_k} is achieved in at most $d_k + d_k \log d_k$ iterations of the while-loop (Lemma 2). As soon as the domination condition is satisfied, agent a_k performs an EQUAL operation (instead of TRIM) on the output allocation of DOMINATION$(R^{d_k+d_k log d_k}, k+1)$. This process produces no residue and the while-loop terminates. Towards this end, agent a_k produces d_k-many equally valued bundles due to `Trim` and `Equal` operations that forms her `Storage`$(k-1, a_k)$ set. This helps in establishing Condition C2 of k-FAIRness for allocation $\mathcal{A}$.

Finally, the count of $d_k + d_k \log d_k$ on the number of rounds of while-loop leads to the desired runtime for our protocol. We will prove that DOMINATION(R, k) outputs a k-FAIR allocation, by showing that all three conditions (in Definition 4) are satisfied (see Lemma 2).

Overall, the final output of DOMINATION$([0,1], 1)$ is a 1-FAIR allocation of the cake $[0,1]$. The run-time of DOMINATION(R, k) and its recursive nature leads to the query complexity n^{2n} for the DOMINATION$([0,1], 2)$.

Notation Guide for Domination(R, k)**:** We say any round t of the while-loop has residue R^t at its beginning. We write $\mathcal{B}^t$ to be the output of DOMINATION$(R^t, k+1)$ in Step 3. Furthermore, $\mathcal{X}^t$ denotes the output of the SELECT procedure (Step 4) and $\mathcal{Y}^t$ denotes the output of the TRIM and EQUAL procedures performed by agent a_k (Steps 10 and 16). We will drop the superscript t whenever it is clear from the context. Finally, we write $\mathcal{A} = (A_1, \ldots, A_n)$ to be the output allocation of DOMINATION(R, k).

Theorem 2. *For cake-division instances with n agents on a* TREE*, there exists a discrete protocol that computes a locally envy-free allocation using at most $O(n^{2n})$ Robertson-Webb queries.*

We begin with a crucial lemma (Lemma 2) which proves that DOMINATION(R, k) returns a k-FAIR allocation after at most $d_k + d_k \log(d_k)$ many runs of the while loop. This property forms the crux of our recursive protocol DOMINATION(R, k).

Lemma 2. *Consider any cake-division instance with n agents on a* TREE *graph. For any $R \subseteq [0,1]$ and $k \in [n]$,* DOMINATION(R, k) *computes a k-*FAIR *allocation in $d_k + d_k \log d_k$ rounds of the while-loop.*

Next, we state three properties in Lemma 3 that are instrumental in proving the above lemma. Its proof is deferred to the full version of the paper [18].

Lemma 3. DOMINATION(R, k) *has following three properties:*

1. *For any $j \notin D(k) \cup p_k$,* `Storage`(k, a_j) = `Storage`$(k-1, a_j)$*. Furthermore, this set remains intact during the entire execution of* DOMINATION(R, k)*. Moreover, for $j = p_k$ we have* `Storage`$(k-1, a_{p_k}) \subseteq$ `Storage`(k, a_{p_k})*.*
2. *For any round t of the while-loop during the protocol* DOMINATION(R, k)*. Then, after $t + d_k \log d_k$ rounds of the while-loop, we obtain $v_{p_k}(R^{t+d_k \log d_k}) \leq c_t$. Here, $c_t := \max_{\ell \in D_k}\{v_{p_k}(B^t_{p_k}) - v_{p_k}(Y^t_\ell)\}$ and bundles Y^t_k for $k \in [\ell_i]$ are*

Recursion step: DOMINATION(R, k) for TREES

Input: A cake-division instance $\mathcal{I}$ on a TREE, a piece $R \subseteq [0,1]$, and an index $k \in \{1, \ldots, n-1\}$.
Output: A k-FAIR allocation of R.

1 Initialize bundles $A_i \leftarrow \emptyset$ for $i \in [n]$ and set a counter $c \leftarrow 0$
2 **while** $R \neq \emptyset$ **do**
3     $\mathcal{B} \leftarrow$ DOMINATION$(R, k+1)$
    ——— `Selection` ———
4     $\mathcal{X} \leftarrow$ SELECT$(a_k, \texttt{Storage}(k, a_{p_k}), d_k)$ `/* Storage set for` $\mathcal{B}$ `*/`
5   Re-index the bundles in $\mathcal{X}$ and $\texttt{Storage}(k, a_{p_k}) \setminus \mathcal{X}$ so that they bear the indices in D_k and $\texttt{Inact}(k, a_{p_k}) \setminus D_k$ respectively
    `/* We can do this because Step 3 ensures that agent` a_{p_k} `is indifferent towards the bundles in Storage`(k, a_{p_k}) `*/`
6   **for** $j \notin D_k$ **do**
7         $A_j \leftarrow A_j \cup B_j$ **if** $\exists\, \ell \in D_k$ *such that* $v_{p_k}(A_{p_k}) - v_{p_k}(A_\ell) \leq v_{p_k}(R)$
            `/* Checking the domination condition for agent` a_{p_k} `*/`
        ——— `Trimming` ———
8   Set $R \leftarrow \emptyset$
9         $(\mathcal{Y}, R) \leftarrow$ TRIM$(a_k, \mathcal{X})$
10   Let $Y_g = \arg\min_{\ell \in D(k)} v_{p_k}(Y_\ell)$ and $w = c \mod d_k + 1$
11         $A_w \leftarrow A_w \cup Y_g$ and $\mathcal{Y} \leftarrow \mathcal{Y} \setminus Y_g$
        `/* Trying to achieve domination on` A_w `for the agent` a_{p_k} `*/`
12   For each $\ell \in D_k \setminus \{w\}$, add one arbitrary piece from $\mathcal{Y}$ to A_ℓ
13         $c \rightarrow c + 1$
14   **else**
            ——— `Equaling` ———
15             $\mathcal{Y} \leftarrow$ EQUAL$(a_k, \mathcal{X})$
16   For each $i \in D_k$, add one arbitrary piece from $\mathcal{Y}$ to A_i
17 **return** Allocation $\mathcal{A} = (A_1, \ldots, A_n)$

obtained after the TRIM *procedure in Step 10 of* DOMINATION(R, k) *protocol performed by agent* a_k *in round* t.

3. *During the execution of* DOMINATION(R, k)*, the difference between the value of agent* a_{p_k} *for her bundle and for the bundle of any agent in the set* $D(k)$ *increases with each round of the while-loop i.e., for any round* t*, we have*

$$v_{p_k}(A^t_{p_k}) - v_{p_k}(A^t_\ell) \leq v_{p_k}(A^{t+1}_{p_k}) - v_{p_k}(A^{t+1}_\ell) \quad \textit{for all } \ell \in D_k$$

where $\mathcal{A}^t$ *and* $\mathcal{A}^{t+1}$ *are the allocations at the end of rounds* t *and* $t+1$.

Proof of Lemma 2: Given any piece $R \subseteq [0,1]$ and $k \in [n]$, we begin by proving that the output allocation, $\mathcal{A} = (A_1, A_2, \ldots, A_n)$ of the DOMINATION(R, k) is k-FAIR. Towards the end, we will prove the desired count on the number of while-loops that suffices to achieve so.

We will proceed via recursion on $k \in [n]$. Recall that DOMINATION(R, n) asks agent a_n to simply divide R into n equal pieces, making it n-FAIR trivially.

Now, let us assume that the claim holds true for $k+1$, and we will prove it for k. That is, in every round of the while-loop, the output allocation $\mathcal{B}$ of Domination$(R, k+1)$ is $(k+1)$-Fair. We will show that Domination(R, k) is k-Fair by proving that its output allocation $\mathcal{A}$ satisfies Conditions C2 and C3 and then finally Condition C1 will follow.

- **Condition C2:** For each round of the while-loop, since $\mathcal{B}$ is $(k+1)$-Fair, we have $v_j(B_j) = v_j(B)$ for all $B \in \texttt{Storage}(k, a_j)$ for all $j \geq k+1$ from Condition C2. For $j \geq k+1$ and $j \neq p_k$, note that Lemma 3 implies that the bundles in the $\texttt{Storage}(k, a_j)$ set remains intact during Domination(R, k). Since $\texttt{Storage}(k, a_j) = \texttt{Storage}(k-1, a_j)$, the desired condition is satisfied for these agents.

Now, for $j = p_k$, the induction hypothesis implies that $v_{p_k}(B_{p_k}) = v_{p_k}(B)$ for all $B \in \texttt{Storage}(k, a_{p_k})$. And Lemma 3 implies that $\texttt{Storage}(k-1, a_{p_k}) \subseteq \texttt{Storage}(k, a_{p_k})$. Hence, we obtain the desired relation.

Finally, let us consider agent a_k. We know that agent a_k selects d_k many bundles from $\texttt{Storage}(k, a_{p_k})$ in each round of the while-loop and performs a `Trim` procedure on this set to make them all of equal value to her. The equaling phase maintains this property, hence establishing Condition C2 of k-Fairness.

- **Condition C3:** Let us consider agents a_k and a_{p_k}. We show that $v_{p_k}(A_{p_k}) \geq v_{p_k}(A)$ for all $A \in \texttt{Storage}(k-1, a_k)$. Towards the end of the protocol Domination(R, k), we know that agent a_k creates her $\texttt{Storage}(k-1, a_k)$ set containing d_k many equally-valued bundles. In each round of the while-loop, a_k selects her d_k-many favorite pieces from the set $\texttt{Storage}(k, a_{p_k})$ (of $(k+1)$-Fair allocation of that round) in Step 4 and performs Trim until a_{p_k} starts dominating her.

At the termination round, say T, of the while-loop, we have (by the domination condition stated in Step 8)

$$v_{p_k}(A_{p_k}^T) - v_{p_k}(A_\ell^T) \geq v_{k+1}(R^T) \text{ for all } \ell \in D(k),$$

where A_ℓ^T denotes the bundle of agent a_ℓ formed at the end of round T of the while-loop. The domination condition implies that the residue R^T is small enough that it does not induce any envy for a_{p_k} even if R^T is fully allocated to any bundle A_ℓ^T for $\ell \in D(k)$. The `Equaling` phase maintains the similar relation, and hence we obtain $v_{p_k}(A_{p_k}) \geq v_{p_k}(A_j)$ for all $j \in D_k$. Since, these A_j's form the $\texttt{Storage}(k-1, a_k)$ set, we obtain the desired relation.

Finally, for any other active child a_ℓ of a_{p_k}, note that Lemma 3 says that the $\texttt{Storage}(k, a_\ell)$ remains intact, i.e., we have $\texttt{Storage}(k-1, a_\ell) = \texttt{Storage}(k, a_\ell)$. Since bundle $B_{p_k} \in \texttt{Storage}(k, a_{p_k})$ is present in $\texttt{Storage}(k-1, a_{p_k})$, Condition C3 follows from the induction hypothesis.

- **Condition C1:** Let us first consider agent a_k. In any round t of the while-loop, she selects her d_k most preferred pieces from $\texttt{Storage}(k, a_{p_k})$ set (corresponding to $\mathcal{B}$). We re-index the bundles such that B_{p_k} is one of the remaining pieces, and that is allocated to agent a_{p_k}. The `Trimming` and `Equaling` phases ensure that we maintain $v_k(A_k) \geq v_k(A_{p_k})$. The fact that a_k does not envy any of her children follows from Condition C2, proved above. Condition C3 can be used to prove that a_{p_k} does not envy non-children agent a_k as well.

Consider an agent a_j such that $j \notin (D_k \cup p_k)$. All that is left is to prove that agent a_j does not envy her neighbours in the output allocation $\mathcal{A}$. This is true since the topological ordering ensures that any agent in the set $D(k)$ has an index that is less than k. Since agent a_j is allocated an intact piece from $\mathcal{B}$ in Steps 6-7, using the induction hypothesis, we obtain that she does not envy her neighbours in the allocation $\mathcal{A}$. This proves that $\mathcal{A}$ satisfies Condition C1 of k-Fairness.

Runtime Analysis: To establish the runtime of Domination(R, k), we will consider the first d_k rounds of the while-loop during its execution. If our protocol terminates before d_k rounds, we are done. If not, observe that Steps 11-12 allocate the smallest piece after trimming (i.e. $\arg\min_{\ell \in D(k)} v_{p_k}(Y_\ell)$) to different bundles in the first d_k rounds. Assume, without loss of generality, bundle A_h is allocated the smallest trimmed piece in round $h \in [d_k]$. Recall the definition of the maximum trimmed value $c_h := \max_{\ell \in D_k}\{v_{p_k}(B^h_{p_k}) - v_{p_k}(X^h_\ell)\}$ for round h. Since agent a_{p_k} gets bundle $B^h_{p_k}$ (which is not trimmed) in this round, we know that $v_{p_k}(A^h_{p_k}) - v_{p_k}(A^h_\ell) \geq c_h$ for all $\ell \in D(k)$. Using this inequality for all $h \leq d_k$ we have,

$$
\begin{aligned}
v_{p_k}(A^{d_k+d_k\log d_k}_{p_k}) - v_{p_k}(A^{d_k+d_k\log d_k}_{\ell}) &\geq v_{p_k}(A^h_{p_k}) - v_{p_k}(A^h_h) && \text{(by Lemma 3)}\\
&\geq c_m && \text{(by Step 12)}\\
&\geq v_{p_k}(R^{d_k+d_k\log d_k}) && \text{(by Lemma 3)}
\end{aligned}
$$

Therefore, after at most $d_k + d_k \log d_k$ rounds, the agent a_k enters the `Equaling` phase and the while-loop terminates to output the final k-Fair allocation. □

Proof of Theorem 2: Note that the k-Fairness of Domination(R, k) ensures that the final output allocation $\mathcal{A}^*$ of Domination$([0, 1], 2)$ is locally envy-free.

We denote the query complexity of Domination(R, k) by T_k for $k \in [n]$; we will prove $T_1 = O(n^{2n})$. Note that, each round of the while-loop during the execution of the protocol Domination(R, k) requires d_k `eval` queries and $d_k - 1$ `cut` queries. Lemma 2 proves that Domination(R, k) outputs a k-Fair allocation in $d_k + d_k \log d_k$ rounds of the while-loop. Now, let us first observe the execution of Domination$(R, 1)$ protocol. It invokes Domination$(R, 2)$ which makes T_2 many queries to output $\mathcal{C}$. The corresponding set (in $\mathcal{C}$) $\texttt{Storage}(1, a_i) = \{C_i\}$ for every agent a_i, except the parent agent of a_1. We have $\texttt{Storage}(1, a_{p_1}) = \{C_1, C_{p_1}\}$, out of which agent a_1 selects her favorite bundle (by making two `eval` queries) and the remaining bundle. That is, we can write $T_1 = T_2 + 2$.

Since the agents are indexed according to the topological order, we have $d_k \leq k$ for all $k \in [n]$. Hence, we will derive the query complexity in terms of k instead of d_k. We prove that $T_k \leq \sum_{j=k}^{n} j \prod_{i=k}^{j}(3i \log i)$ using induction on k. For the base case $k = n$, we know that $T_n = n \leq 3n^2 \log(n)$. Assuming that the bound holds for T_{k+1} and writing T_k in terms of T_{k+1} we have,

$$
\begin{aligned}
T_k &\leq (d_k + d_k \log d_k)T_{k+1} + d_k(d_k + d_k \log d_k) \leq (k + k \log k)T_{k+1} + 2k^2 \log k\\
&\leq 3k \log k \sum_{j=k+1}^{n} j \prod_{i=k+1}^{j}(3i \log i) + 2k^2 \log k = \sum_{j=k+1}^{n} j \prod_{i=k}^{j}(3i \log i) + 2k^2 \log k
\end{aligned}
$$

$$< \sum_{j=k+1}^{n} j \prod_{i=k}^{j} (3i \log i) + k(3k \log k) \leq \sum_{j=k}^{n} j \prod_{i=k}^{j} (3i \log i). \quad (4)$$

Let us now finally bound T_2 using the bound described in equation 4). We obtain $T_2 \leq \sum_{j=}^{n} j \prod_{i=2}^{j} (3i \log i)$. Let us now denote $h_j = j \prod_{i=2}^{j} (3i \log(i)) = j \cdot 3^j \cdot j! \prod_{i=2}^{j} \log i$. Note that, for all $2 \leq j < n$ we have $2h_j < h_{j+1}$, and hence, $\sum_{j=2}^{n-1} h_j < h_n$. We have $T_2 \leq \sum_{j=2}^{n} j \prod_{i=2}^{j} (3i \log i) = \sum_{j=2}^{n} h_j \leq 2h_n = 2n \cdot 3^n \cdot n!(\log n)^n$. Using Stirling's approximation, we obtain $T_2 = O(n^{2n})$. □

5 Discussion and Future Directions

In this paper, we studied the open problems stated in [1,7] by (a) developing a discrete and bounded protocol for local envy-freeness for n agents on Tree graphs with a single-exponential query complexity, and (b) constructing a query-efficient protocol for computing locally envy-free allocations among n agents on Depth2Tree. We believe that exploring the complexity of envy-free cake division with graphical constraints will give us novel insights and help us understand the hidden bottlenecks in the query complexity of the general problem.

Our work raises an interesting question of developing query efficient algorithms for finding locally envy-free allocations for fixed parameters such as arboricity or the tree-width of the graph. A second interesting research direction is to study trees with a constant depth and check if can we develop query-efficient protocols for these graphs. Developing efficient protocols for graphs such as a cycle or a bipartite graph is also an interesting future direction.

Acknowledgments: Xin was supported in part at the Technion Israel Institute of Technology by an Aly Kaufman Fellowship. Ganesh was supported by Department of Science and Technology, India under grant CRG/2022/007927. The authors thank Siddharth Barman, Ioannis Caragiannis, and Amik Raj Behera for their helpful comments.

References

1. Abebe, R., Kleinberg, J., Parkes, D.C.: Fair division via social comparison. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems (AAMAS). p. 281–289 (2017)
2. Amanatidis, G., Christodoulou, G., Fearnley, J., Markakis, E., Psomas, C., Vakaliou, E.: An improved envy-free cake cutting protocol for four agents. In: Proceedings of the 11th International Symposium on Algorithmic Game Theory (SAGT). pp. 87–99 (2018)
3. Arunachaleswaran, E.R., Barman, S., Kumar, R., Rathi, N.: Fair and efficient cake division with connected pieces. In: Proceedings of the International Conference on Web and Internet Economics (WINE). pp. 57–70 (2019)
4. Aziz, H., Mackenzie, S.: A discrete and bounded envy-free cake cutting protocol for any number of agents. In: Proceedings of the 57th Annual Symposium on Foundations of Computer Science (FOCS). pp. 416–427 (2016)

5. 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. p. 454–464. (STOC) (2016)
6. Barman, S., Rathi, N.: Fair cake division under monotone likelihood ratios. Mathematics of Operations Research pp. 1875–1903 (2022)
7. Bei, X., Qiao, Y., Zhang, S.: Networked fairness in cake cutting. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence, (IJCAI). pp. 3632–3638 (2017)
8. Bei, X., Sun, X., Wu, H., Zhang, J., Zhang, Z., Zi, W.: Cake cutting on graphs: a discrete and bounded proportional protocol. In: Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA). pp. 2114–2123 (2020)
9. Brams, S.J., Taylor, A.D.: Fair Division: From cake-cutting to dispute resolution. Cambridge University Press (1996)
10. Brandt, F., Conitzer, V., Endriss, U., Lang, J., Procaccia, A.D.: Handbook of computational social choice. Cambridge University Press (2016)
11. Bredereck, R., Kaczmarczyk, A., Niedermeier, R.: Envy-free allocations respecting social networks. Artificial Intelligence **305**, 103664 (2022)
12. Budish, E.: The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. Journal of Political Economy pp. 1061–1103 (2011)
13. Deng, X., Qi, Q., Saberi, A.: Algorithmic solutions for envy-free cake cutting. Operations Research **60**(6), 1461–1476 (2012)
14. Dubins, L.E., Spanier, E.H.: How to cut a cake fairly. The American Mathematical Monthly **68**(1P1), 1–17 (1961)
15. Edward Su, F.: Rental harmony: Sperner's lemma in fair division. The American mathematical monthly **106**(10), 930–942 (1999)
16. Etkin, R., Parekh, A., Tse, D.: Spectrum sharing for unlicensed bands. IEEE Journal on selected areas in communications **25**(3), 517–528 (2007)
17. Foley, D.K.: Resource Allocation and the Public Sector, vol. 7:45-98. Yale Economic Essays (1966)
18. Ghalme, G., Huang, X., Machino, Y., Rathi, N.: A discrete and bounded locally envy-free cake cutting protocol on trees. arXiv preprint arXiv:2211.06458 (2022)
19. Goldman, J., Procaccia, A.D.: Spliddit: Unleashing fair division algorithms. ACM SIGecom Exchanges **13**(2), 41–46 (2015)
20. Kurokawa, D., Lai, J., Procaccia, A.: How to cut a cake before the party ends. In: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI). pp. 555–561 (2013)
21. Moulin, H.: Fair division and collective welfare. MIT press (2004)
22. Procaccia, A.D.: Thou shalt covet thy neighbor's cake. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI). p. 239–244 (2009)
23. Robertson, J., Webb, W.: Cake-cutting algorithms: Be fair if you can. AK Peters/CRC Press (1998)
24. Steinhaus, H.: The problem of fair division. Econometrica **16**, 101–104 (1948)
25. Stromquist, W.: How to cut a cake fairly. The American Mathematical Monthly **87**(8), 640–644 (1980)
26. Stromquist, W.: Envy-free cake divisions cannot be found by finite protocols. Electronic Journal of Combinatorics **15**(1) (2008)
27. Tucker-Foltz, J.: Thou shalt covet the average of thy neighbors' cakes. Information Processing Letters **180**, 106341 (2023)