Envy-free Chore Division for An Arbitrary Number of Agents*

Sina Dehghani †‡ Alireza Farhadi †‡ MohammadTaghi HajiAghayi †‡ Hadi Yami †‡

Abstract

Chore division, introduced by Gardner in 1970s [10], is the problem of fairly dividing a chore among n different agents. In particular, in an envy-free chore division, we would like to divide a negatively valued heterogeneous object among a number of agents who have different valuations for different parts of the object, such that no agent envies another agent. It is the dual variant of the celebrated cake cutting problem, in which we would like to divide a desirable object among agents. There has been an extensive amount of study and effort to design bounded and envy-free protocols/algorithms for fair division of chores and goods, such that envy-free cake cutting became one of the most important open problems in 20-th century mathematics according to Garfunkel [11]. However, despite persistent efforts, due to delicate nature of the problem, there was no bounded protocol known for cake cutting even among four agents, until the breakthrough of Aziz and Mackenzie [2], which provided the first discrete and bounded envy-free protocol for cake cutting for four agents. Afterward, Aziz and Mackenzie [3], generalized their work and provided an envy-free cake cutting protocol for any number of agents to settle a significant and long-standing open problem. However, there is much less known for chore division. Unfortunately, there is no general method known to apply cake cutting techniques to chore division. Thus, it remained an open problem to find a discrete and bounded envy-free chore division protocol even for four agents.

In this paper, we provide the first discrete and bounded envy-free protocol for chore division for an arbitrary number of agents. We produce major and powerful tools for designing protocols for the fair division of negatively valued objects. These tools are based on structural results and important observations. In general, we believe these structures and techniques may be useful not only in chore division but also in other fairness problems. Interestingly, we show that applying these techniques simplifies Core Protocol provided in Aziz and Mackenzie [3].

1 Introduction

The *chore division* problem is the problem of fairly dividing an object deemed undesirable among a number of agents. The object is possibly heterogeneous, and hence agents may have different valuations for different parts of the object. Chore division was first introduced by Gardner [10] in 1970s, and is the dual problem of the celebrated *cake cutting* problem. In cake cutting, we would like to fairly divide a good (such as a cake) for which everyone has a positive valuation. In some sense, chore division is a minimization problem while cake cutting is a maximization problem. Recently Aziz and Mackenzie [2] provided a bounded *envy-free* protocol for 4-person cake cutting, and later on a bounded envy-free protocol for n-person cake cutting [3]. Chore division or cake-cutting with negative utilities is less explored and much less is known about it. In this paper, we provide the first *discrete and bounded envy-free chore division protocol for any number of agents.*

The fair cake-cutting problem was introduced in the 1940s. There are different ways one can define *fairness*. Initially, *proportional division* was studied. An allocation is *proportional* if everyone receives at least a $\frac{1}{n}$ fraction of the cake according to his/her valuation. Proportional division was solved soon in 1950 [23]. A stronger criterion of envy-freeness was proposed by George Gamow and Marvin Stern in 1950s, which is, *no one envies another*. In other words, each agent receives a part he thinks is the largest part.[1]. The envy-free cake cutting problem became "one of the most important open problems in 20th-century mathematics" according to Garfunkel [11].

For the case of two agents, the *"I cut you choose"* protocol simply provides an envy-free allocation for both cake cutting and chore division. However, the problem is highly more complicated for more agents. In general, since the valuations of agents for different

*The omitted proofs can be found in the full version of this paper.

†University of Maryland. Email: Sina.Dehghani@gmail.com,{farhadi,hajiagha}@cs.umd.edu, hyami@umd.edu

‡Supported in part by NSF CAREER award CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF-1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant.

[1] It is easy to see an envy-free allocation is also proportional

parts of the object may be complex, the standard is to assume a query access model for evaluations. We can ask an agent its value for a part of the object, and also ask an agent to trim the object up to a certain value. For the case of three agents, Selfridge and Conway independently found an envy-free protocol for cake cutting. Oskui (see [20]) provided a solution for 3-person chore division, which is similar to Selfridge-Conway procedure for cake cutting, but is more complicated and needs 9 cuts instead of 5. Finding a finite protocol for cake cutting with more than three agents remained an open problem for a long time until [6] presented a *finite* envy-free protocol for cake cutting for any number of agents in 1995. Although this was a breakthrough in the field, their protocol is finite but *unbounded*, i.e., it does not guarantee *any bound* on the number of queries and even the number of cuts. Later Peterson and Su Peterson and Su [16] provided an unbounded envy-free protocol for chore division. Brams et al. [5] and Saberi and Wang [21] gave "moving-knife" protocols for cake-cutting for four and five agents. Peterson and Su [15] gave a moving-knife procedure for 4-person chore division. A moving-knife procedure involves one or more agents moving knives simultaneously with some restrictions until one agent calls "stop". Although moving-knife procedures are more than existence theorems, "a moving knife protocol is certainly less than an effective procedure in the algorithmic sense" according to [12]. That is because the continuous movement of a knife cannot be captured by any finite protocol.

Having a bounded envy-free protocol even for four agents remained an important open problem [4, 6, 7, 8, 9, 13, 14, 18, 19, 20, 21, 22]. The unboundedness of cake cutting protocols was mentioned as a "serious flaw" [19], and finding a bounded protocols was highlighted as "the central open problem in the field of cake-cutting" [14] and "one of the most important open problems in the field" [21]. Brams and Taylor [6] were aware of their protocols drawback and explicitly mentioned "even for $n = 4$, the development of finite bounded envy-free cake cutting protocols still appears to be out of reach and a big challenge for the future". Finally, the prominent work of Aziz and Mackenzie [2] provided a bounded envy-free cake cutting protocol for four agents. Later they generalized their work and provided an envy-free cake cutting protocol for *any* number of agents to settle a major and long-standing open problem. However, it remained an open problem to find a bounded envy-free chore division protocol even for $n = 4$. In this paper, we provide the first discrete and unbounded envy-free protocol for chore division among any number of agents.

1.1 Prelimiaries In chore division, we are asked to partition a given chore R among n agents. Let $A = \{a_1, \ldots, a_n\}$ be the set of agents, and $C_a(P)$ denote the cost of some piece $P \subseteq C$ for agent a. w.l.o.g we assume that For every agent a, the cost of the whole chore is 1, i.e., $c_a(C) = 1$. An envy-free partition is a partition of R into n pieces $P_1, \ldots, P_n$ and assigning them to the agents accordingly such that for every two agents a and b, $C_a(P_a) \leq C_a(P_b)$, where P_a and P_b denote the pieces assigned to agents a and b respectively.

For any protocol, we use the standard Robertson-Webb model [20]. In Robertson-Webb model, the chore is modeled as an interval $R = [0, 1]$. We have absolutely no knowledge about the agents' cost functions in advance, except that the functions are defined on sub-intervals of $[0, 1]$, non-negative, additive, divisible, and normalized. Therefore every information is obtained via queries. The complexity of a protocol is defined by the number of queries it makes. There are two types of information queries:

- $Trim_a(\alpha)$: given a cost value $0 \leq \alpha \leq 1$, agent a returns an $0 \leq x \leq 1$, such that his cost for interval $[0, x]$ equals α.
- $Eval_a(x)$: returns the cost value of interval $[0, x]$ for agent a.

In this paper, we distinguish between cutting and trimming of a piece. Cutting a piece P refers to dividing P into two pieces, but in trimming we only find a subinterval in P and do not cut the piece. Note that in this paper a piece P is not necessarily an interval, but a union of intervals, since we may cut and join pieces. Although $Eval$ and $Trim$ queries are defined on intervals, whenever we cut a piece we maintain the cost of the new pieces. Thus we can translate a query on a piece to a query on the interval $[0, 1]$.

1.2 Results and Techniques Our main result is a discrete and bounded envy-free protocol for dividing a chore among n agents. Many techniques have been proposed for envy-free cake cutting. Aziz and Mackenzie [3] provide a bright and powerful framework to obtain a bounded and envy-free protocol for cake cutting among n agents. However the components of their framework and their protocols do not work for chore division. The protocols for positive valuations are not usually applicable for negative valuations, and "in general there are no reductions from allocation to chores to goods or vice versa"[1]. To solve the chore variant of the problem, we borrow the general idea of their framework, but we have to provide novel techniques and structural results and also rebuild their framework's components. These new techniques and structures not only deliver powerful

tools for designing chore division protocols, but also are useful in cake cutting.

In the following, we present the very high-level concepts and techniques used in this paper. The basic idea is to use an inductive algorithm. More precisely we use induction on the number of agents and try to divide a chore only among a subset of the agents.

Initially, we need an envy-free protocol which partially divides a chore among the agents. The protocol does not necessarily allocate the whole chore, but roughly speaking assigns a fraction of the chore, maintaining the envy-freeness. The protocol has other plausible features to be mentioned later.

Having a partial allocation, we use the concept of *irrevocable advantage (dominance)*. It is the key of many fair allocation protocols [2, 3, 6, 15, 16, 21]. Assume that the partial allocation is envy-free and we have a remaining or unallocated chore R. We say an agent a has an irrevocable advantage to another agent b or a dominates b, if a thinks she is assigned much less chore than b, such that she may not envy b even if we assign the whole R to her. In other words, $C_a(P_b) - C_a(P_a) \geq C_a(R)$, where P_a and P_b are the pieces allocated to a and b respectively in the partial allocation. We use a similar but weaker notion of *significant advantage*. Agent a has a significant advantage over b if P_a is much more desirable than P_b to a with respect to the remaining chore, or more precisely $C_a(P_b) - C_a(P_a) \geq \alpha \times C_a(R)$, where α is a constant to be defined later. Importantly we show that significant advantage and irrevocable advantage are in some sense equivalent. If agent a has an irrevocable advantage over b, then her advantage is significant as well. On the other hand if agent a has a significant advantage over agent b, using some partial allocation protocols we make R small enough for agent a to make the advantage irrevocable.

Assume that we have a partial envy-free allocation. If there exists a set of agents $S \subset A$, such that each agent in S has irrevocable advantage to every agent in $A \setminus S$, we can leave $A \setminus S$ unchanged, and assign the remaining chore inductively to S. Thus the main goal of our protocol is to make a set of agents have significant/irrevocable advantage over the rest of the agents.

The other very useful concept, introduced by Aziz and Mackenzie [3], is the notion of *snapshots*. Recall that we have a partial allocation protocol. Every time we may partially allocate the remaining chore to the agents. Each of these partial allocations is called a *snapshot*. The chore assigned to each agent is the union of her assigned chores in all the snapshots. A critical thing about snapshots is that we can use an agent's advantage in one snapshot to compensate her for modifications in other snapshots. Basically, if agent a has a lot of advantage over b in one snapshot, we can for example assign some of b's chore to a in some other snapshot. Also note that, as long as every snapshot is envy-free, if an agent a has irrevocable advantage to agent b, then she also has irrevocable advantage in total. Thus we can focus on one snapshot and deliver irrevocable advantage among some agents in that single snapshot. Then we can use other snapshots for having irrevocable advantage among other agents. Another very handy use of snapshots is that we may have as many of them as we need. Then we can concentrate on a set of similar snapshots. More precisely, in a snapshot every agent can order the other agents based on how much is their value for her allocated piece. [2] define two snapshots *isomorphic* if, roughly speaking, those orderings of the agents are exactly the same. Here we need a stronger notion of isomorphism. First, we define a *mask* of a snapshot, which somehow codes the significance of agents' advantages. We say two snapshots are isomorphic if each agent orders the other agents exactly the same and also their masks are the same. Having isomorphic snapshots, we can modify the allocated pieces easier, and thus we construct as many snapshots to be able to have a large enough set of isomorphic snapshots, using pigeon hole principle. We initially call this set of snapshots the *working set*. We set aside the other snapshots and only modify the working set.

The other useful concept that we introduce is a *matching*. A set of trimmed pieces and agents have a matching if we can match every trimmed piece to an agent such that the allocation is envy-free. We use this extra information about pieces to obtain more structural protocols. We show that if we have a matching we can define *monotone* protocols, which means we may only make the trimmed pieces larger, obtaining an envy-free allocation. We also use matching in the *Sub Core* protocol, in which we put a lower bound on the trims of pieces and try to trim the pieces and guarantee to maintain a matching.

Now we describe a technical overview of the main protocols. As aforementioned we need a partial envy-free allocation protocol called the *core* protocol. The Core protocol is the main and most fundamental protocol of our algorithm. The Core protocol has to have the following properties.

- Assigns each agent a piece such that no agent envies another agent;
- Assigns at least a $\frac{1}{n}$ fraction of the chore in one agent's point of view;
- Most importantly, given a specific agent, guaran-

tees that this agent has significant advantage to another agent in this allocation.

The Core protocol is the "engine" of Aziz and Mackenzie [2, 3]'s protocol for cake cutting, but unfortunately their protocols are not applicable for the chore division. Instead we design a much simpler Core protocol. Although our Core protocol is very simple, its proof is based on a much more complicated infinite protocol which guarantees the existence of the desired allocation. The basic idea of a Core protocol is as follows. We select a cutter agent that divides the chore into n equal pieces. Note that the pieces are not necessarily equal to other agents. Then we try to match each agent to one piece such that every agent receives a part of its matched piece, but at least one agent may be given a whole piece. Thus, at least $\frac{1}{n}$ fraction of the cake is allocated in the cutter's point of view. Also, the cutter receives some considerable advantage to the agent who has been given a whole piece. The heart of our Core protocol is the following structural lemma, which is the restatement of Lemma 3.8.

LEMMA 1.1. *Given n pieces and n different agents, there exists an allocation of pieces to agents such that a whole piece is allocated to one agent, and a trim of each piece is allocated to exactly one agent, if and only if, there exists an ordering of the agents and an ordering of the pieces such that the following protocol provides an envy-free allocation. Agents receive their pieces one by one. The first agent receives the first piece. The i-th agent trims the i-th piece in such a way that she receives the largest part of it without envying the first $i-1$ agents. In other words she considers the first $i-1$ allocated pieces, if her cost for any of those pieces is less than her cost for the i-th piece, she trims the i-th piece to make it equal to that piece.*

Note that in such a protocol, the i-th agent may not envy the first $i-1$ agents, but some of the first $i-1$ agents may envy the i-th agent. Roughly speaking, this lemma shows that if there exists some "core-like" allocation of some pieces to agents, there exist an ordering of both agents and pieces such that the first $i-1$ agents also do not envy the i-th agent, using the aforementioned protocol. Thus, if there exists such allocation, we can try every ordering of agents and pieces to find an envy-free allocation using that simple protocol. Interestingly, we design a protocol which is even *infinite* but outputs a core-like allocation. However, knowing that there exists such a protocol is sufficient to be able to design a much simpler Core protocol.

Another important aspect of this structural result is that it also holds for cake cutting. Aziz and Mackenzie [2] provide a relatively complicated Core protocol. Using our structure, we may design a much simpler protocol. Since they provide a Core protocol, it implies that there exists a core-like allocation, and thus our simple protocol also works for cake cutting.

The other important component of our protocol is the *Permutation* protocol. Assume that there is a set $S \subset A$ of agents, such that every agent in S has significant advantage to some agent $a \in A$ in a set of snapshots. If we could exchange the piece allocated to a with the allocated piece of some other agent $b \notin S$ in some snapshot, then every agent in S also has significant advantage to b. In Permutation we try to find such set S, that has significant advantage to a, and then somehow move the a's piece among every agent not in S. Therefore every agent in S has significant advantage to every agent in $A \setminus S$, and we can do the chore division inductively as we discussed. For exchanging the agents' pieces we find a chain of agents, $a_1, a_2, \ldots, a_k$, such that a_i receives a_{i-1}'s piece and a_1 receives a_k's piece. Since each snapshot is envy-free after changing the pieces, agents $a_1, \ldots, a_k$ may envy each other or other agents. Thus we modify many other snapshots to guarantee envy-freeness. Aziz and Mackenzie [3] also have a Permutation protocol. The key difference between our Permutation protocol is that in cake cutting we can add a piece of cake from R to a piece that is assigned to an agent, such that another agent accepts to receive it. However in chore division we have to remove a part of chore from a piece to be able to assign it to some other agent. The difference is huge and makes the Permutation much more subtle because of the two following reasons. First the part that we remove from a piece assigned to an agent goes back to the remaining chore, or R. Since R becomes larger, the significance of the advantages, which are defined based on R may change. Second, since the pieces that we want to remove are already allocated, it is not easy to divide them between agents, or remove similar pieces from other agents.

Moreover, we make use of two previously known fair division protocols. The protocols are used as infinite protocols for cake cutting, but we show that one can use them as powerful tools for bounded protocols as well. The first protocol is the *Near-exact* protocol introduced by Pikhurko [17]. In the Near-exact protocol, given a chore R, n agents, an integer m, and a real number $\epsilon > 0$, we divide E into m pieces, such that for each agent a and piece P, $|C_a(P) - \frac{1}{m}C_a(R)| \leq \epsilon C_a(R)$. In other words the pieces have almost equal costs for the agents. We also show how one can use the Near-exact protocol to improve Aziz and Mackenzie [3]. The other protocol is the *Oblige* protocol, first used by

Peterson and Su [16]. In Oblige protocol we partition the chore into 2^{n+1} pieces, and output a partial envy-free allocation such that every agent is assigned at least one of the pieces completely. The combination of these protocols is used in our Discrepancy protocol, described below.

Another important component of our protocol is the Discrepancy protocol. We use the Discrepancy protocol when we have a piece P that is very costly for a set of agents S and R is relatively small, and in the contrary the rest of agents think R costs much more than P. Thus we use a combination of Near-exact and Oblige protocols to divide R among S and P among the rest of the agents, such that no agent envies another agent. In this way we may inductively divide the chore among smaller set of agents.

2 Main Protocol

Main Protocol is responsible for allocating the whole chore among the agents in an envy-free manner. It first makes a set of agents dominant to the others and then allocates the remaining chore to a smaller number of agents. Main Protocol achieves this goal by using two other protocols, Core Protocol, and Permutation Protocol. As we mentioned in Introduction, in Core Protocol we are Given an agent a as the cutter who divides the chore into n equally preferred pieces, and then the protocol partially allocates the chore to the agents such that at least one piece is completely allocated and each agent gets part of a single piece. Permutation Protocol gets a partial allocation of the chore and makes a set of agents dominant to others by slightly changing the allocation.

In the beginning, Main Protocol calls Core Protocol many times, each time on the remaining chore to create a large number of partial allocations. We call each of these partial allocations a *snapshot*.

Algorithm 1: Main Protocol

Data: List of agents $A = \{a_1, a_2, \ldots, a_n\}$ and chore R
if $n = 1$ **then**
  allocate the whole chore to agent a_1 ;
  return the allocation;
else if $n = 2$ **then**
  Run cut and choose procedure for agents a_1 and a_2 and chore R ;
  return the allocation;
else
  for $i = 1 to IS_n \times n^{n^{n^n}}$ **do**
    Run Core Protocol(a_1, A, R) to create snapshot s_i and to update the remaining chore;
  for $i = 1 to IS_n \times n^{n^{n^n}}$ **do**
    for *every pair of agent a and b such that* $Adv^{s_i}_{a,b}$ *is not significant* **do**
      Ask agent a to place a trim on s^b_i to make it equal to s^a_i ;
  while *there exists a snapshot* s_i *and pair of agents a and b such that* $C_a(R)(\frac{1}{2^{2n}}) \leq Adv^{s_i}_{a,b} \leq 2^{2n} C_a(R)$ *or* $C_a(R)(\frac{1}{2^{2n}}) \leq C_a(e^{s_i,b}_j) \leq 2^{2n} C_a(R)$ *for some* j **do**
    Run Core Protocol(a, A, R);
    if *an agent c has a significant advantage over agent d in a snapshot* s' **then**
      Remove the trim of agent c from s'^d ;
  if *there exists a set of agents* $B \subset A$ *such that every agent in B has a significant advantage over every other agent in* $A \setminus B$ **then**
    for *each agent* a_i **do**
      Call Core Protocol(a_i, A, R) ;
    Call Main Protocol (B,R) ;
    return the allocation ;
  Find set S of isomorphism snapshots such that $|S| = IS_n$;
  Run Permutation Protocol(C, A, R);
  Let B be the set of agents returned by Permutation Protocol;
  for *each agent* a_i **do**
    Call Core Protocol(a_i, A, R) ;
  Call Main Protocol(B,R) ;
  return the allocation ;

Definition 1. *A* ***snapshot*** *s is a partial envy-free allocation returned by Core Protocol. We use s^a to denote the allocated piece to agent a.*

After generating many snapshots, Main Protocol finds a set of similar snapshots and makes some slight changes on these snapshots in Permutation Protocol. Each time we call Core protocol, we get an envy-free partial allocation of the chore. In each of these snapshots, each agent thinks that the cost of her piece is less than the cost of the others'. In particular, considering a snapshot s and agents a and b, since the partial allocation obtained by Core Protocol is envy-free, agent a thinks that the cost of piece s^a is not greater than cost of s^b. We define the *advantage of agent a over agent b* in this snapshot, the amount of chore that a thinks that she got less than agent b, i.e:

$$Adv^s_{a,b} = C_a(s^b) - C_a(s^a)$$

If the advantage that agent a has over b is greater than the cost of the residual chore, agent a does not envy b, no matter how the residual chore will be allocated among the agents. In this case, we say that agent b is dominated by agent a. In Particular agent a dominates agent b in the partial allocation s if $Adv^s_{a,b} \geq C_a(R)$. Since in Core Protocol the cutter cuts the chore into n equal pieces according to her own perspective and the protocol allocates at least one piece completely, the cost of the residual chore is at most $\frac{n-1}{n}$ for the cutter. In Permutation Protocol, we modify a set of

similar snapshots such that if we reduce the size of chore by calling Core Protocol nB_n times with each agent as the cutter B_n times where $B_n = n^n$, then we can find set of agents B such that each agent in A dominates every other agent in $A \setminus B$. Therefore, if the advantage of agent a over another agent b is at least $C_a(R) \times (\frac{n-1}{n})^{B_n}$ during Permutation Protocol, this agent will dominate agent b after reducing the size of the chore.

We define a value to be *significant* for agent a if it is at least $C_a(R) \times (\frac{n-1}{n})^{B_n}$ and otherwise *insignificant*. Here, a key idea is that if agent a has a significant advantage over agent b, we can reduce the size of the remaining chore such that a dominates b. In Permutation Protocol, we mainly try to modify snapshots such that it gives a significant advantage to a set of agents over all other agents. Since a significant value could become insignificant or vice versa by slightly modifying the residual chore or allocated pieces, we need enlarge the gap between significant and insignificant values to make sure that a significant value remains significant if we only slightly modify the allocated pieces and R. To this end, we define very significant and very insignificant values as follows:

DEFINITION 2. *A value v is **very significant** for an agent if it is at least 2^{2n} times the cost of R in her perspective.*
*A value v is **very insignificant** for an agent if R costs at least 2^{2n} times more than v.*

Aziz and Mackenzie [3] show that we can enlarge the gap between significant and insignificant values using a bounded number of queries by calling Core Protocol many times.

In the beginning of Main Protocol, we run Core Protocol $IS_n \times n^{n^{n^n}}$ times where $IS_n = n^{n^n}$. Our goal is to find a set of IS_n similar snapshots. In each run, we set the first agent as the cutter and partially allocate the residual chore between agents. Let s_i be the snapshot generated in the i^{th} call of Core Protocol. The following claim shows that in each snapshot the cutter has a significant advantage over some other agent.

CLAIM 1. *In each snapshot returned by Core Protocol, the cutter has a significant advantage over at least one other agent.*

After generating snapshots, in each snapshot s, for every agent a, we ask a to place a trim on any piece other than s_a to make it equal to her piece if the cost of this piece is not significantly larger than s_a in her perspective. Main Protocol passes these trim lines to Permutation Protocol which uses the trim lines to modify the allocated pieces and make them desirable for other agents, and then exchanges the pieces between the agents. An important observation is that if in snapshot s an agent a has a significant advantage over some other agent b, and we give s_b to some other agent c while preserving the envy-freeness, then a receives a significant advantage over c in s.

We use $t_1^{s,a}, t_2^{s,a}, \cdots, t_{l_{s,a}}^{s,a}$ to denote the trim lines from right to left on the piece s_a where s is an arbitrary snapshot and a is an arbitrary agent, and $l_{s,a}$ is the number of agents with a trim on this piece. In the same way, we denote the agents with a trim on this piece from right to left by $d_1^{s,a}, d_2^{s,a}, \cdots, d_{l_{s,a}}^{s,a}$. Moreover, we can partition each piece based on the trim lines. We use $e_1^{s,a}, e_2^{s,a}, \cdots, e_{l_{s,a}}^{s,a}$ to partition s_a, where $e_i^{s,a}$ is a part of s_a between two consecutive trims such that the left trim is $t_i^{s,a}$.

Permutation Protocol detaches some part of the pieces from the trim lines. We want the cost of all detached pieces be very small for all the agents, so that very significant advantages remain significant after this procedure. To this end, we make sure that every $e_i^{s,a}$ costs either very significant or very insignificant for all the agents. For this purpose, while there is an agent who thinks at least one part is neither very significant or very insignificant, we keep reducing the size of the residual chore for this agent by calling Core Protocol. After that for every part $e_i^{s,a}$, we define *mask* of this piece or $mask_i^{s,a}$ to be the set of agents who think this part costs very significantly. After that, if we find a part $e_i^{s,a}$ which costs very significant to agent b, and trim line of this agent lies on the left of $t_i^{s^a}$, we can say that agent b receives a very significant advantage over agent a in this snapshot and removes the trim of this agent from s^a.

The set of snapshots given to Permutation Protocol should have very similar properties. In particular, the protocol needs that the order of trims on each piece be the same between different snapshots and every part has the same mask in all the snapshots.

DEFINITION 3. *We call two snapshots s and s' **isomorphic** if :*

- *For every pieces s_a and s'_a, they have the same number of trims on them and order of agents with a trim on these pieces be the same in both snapshots.*
- *For every part $e_i^{s,a}$ and $e_i^{s',a}$, the mask of these parts*

be the same.

In the following lemma, we show that if we generate at least $IS_n \times n^{n^{n^n}}$ snapshots, then we can find at least IS_n isomorphic snapshots. Main Protocol finds these isomorphic snapshots and gives them to Permutation Protocol.

LEMMA 2.1. *Every set of $IS_n \times n^{n^{n^n}}$ has at least IS_n isomorphic snapshots.*

3 Core Protocol

Aziz and Mackenzie in [3] present a Core Protocol as the core engine of their discrete and bounded algorithm for the cake cutting problem. In each call of Core Protocol, they allocate some cake from the residue to all the agents in an envy-free manner. By each call of this protocol, they make the remaining cake smaller, but there is no guarantee that calling this protocol for bounded times suffices to allocate all the cake in an envy-free manner. Nonetheless, they use this protocol several times in different parts of their main algorithm.

In the chore division problem, we have a Core Protocol, Algorithm 2, for allocating additional chore from the residue to all the agents in an envy-free manner. Our Core Protocol works as follows: First we ask the specified cutter to cut the chore into n equal pieces $p_1, p_2, \ldots, p_n$ according to her own perspective. Then, for each ordering of the agents and each ordering of the pieces, we make a new allocation of the pieces to the agents. In the new allocation, agents receive their pieces one by one. The first agent receives the first piece. The i^{th} agent trims the i^{th} piece in such a way to equalize it with her most preferred piece among the first $i-1$ allocated pieces (we consider the cost value of each piece from its leftmost side to its trim.) If this allocation be envy-free we return the allocation. In Lemma 3.8, we guarantee an ordering of agents and ordering of pieces exists such that the protocol returns an envy-free allocation. In Subsections 3.1 through 3.7, we provide another core protocol, which is not bounded but we use it to guarantee such an ordering of the agents and the pieces exists. For this reason, we call the new core protocol *Existential Core Protocol*, Algorithm 3.

3.1 Existential Core Protocol We call our Existential Core Protocol on set of agents A with one specified cutter, and unallocated chore R. In the first step of the protocol we ask the cutter to cut the chore into n equal pieces $p_1, p_2, \ldots, p_n$ according to her own perspective. From now, we work on these n pieces, and we frequently ask the agents to make trims on them. In different steps of the algorithm, we may have many trims on each piece, but we have one specific trim that we call it the *main trim*. We may change the position of the main trim on a piece, but we always have exactly one main trim on each piece. As we mentioned before, our Existential Core Protocol does not necessarily allocate whole of the chore to the agents, and it finally allocates each of these pieces from their leftmost side to their main trim. Initially the main trim of each piece is on its rightmost side, and we change their place frequently during the algorithm. In each step of the algorithm we may allocate a piece up to its main trim to only one agent. It is very crucial to note that, in this section, when we say we allocate a piece to an agent we mean that it is allocated from its leftmost side to its main trim. Also, when we ask the cost value of a piece from a specific agent, she reports her cost value from the leftmost side of the piece to its main trim.

Algorithm 2: Core Protocol

Data: Agent set $A = \langle a_1, a_2, \ldots, a_n \rangle$, specified cutter $a_{cutter} \in A$, and unallocated chore R

Specified cutter a_{cutter} divides the chore into n equal pieces according to her own perspective;
Define $p_1, p_2, \ldots, p_n$ the pieces that we have after the division of a_{cutter};
for *each permutation* $\langle a'_1, a'_2, \ldots, a'_n \rangle$ *of the agents* **do**
  for *each permutation* $\langle p_{a'_1}, p_{a'_2}, \ldots, p_{a'_n} \rangle$ *of the pieces* **do**
    Allocate $p_{a'_1}$ to a'_1 completely;
    for *i from 2 to n* **do**
      Ask a'_i to trim $p_{a'_i}$ to equalize it with her most preferred piece among the first $i-1$ allocated pieces (we consider the cost value of each piece from its leftmost side to its trim.);
      Allocate $p_{a'_i}$, from its leftmost side to its trim, to a'_i;
    if *none of the agents envies to another agent* **then**
      return the envy-free partial allocation (at least one of the pieces has been completely allocated) and the unallocated chore;
    Ignore the previous trims and deallocate the allocated pieces;

In Algorithm 3, after the cutter cuts the chore, we run the Separated Chore Core Protocol on all the agents and all the pieces with their main trims. The Separated Chore Core Protocol receives n pieces of the chore with their main trims and a set of n agents, and it returns an envy-free partial allocation of the pieces to the agents. The properties of Separated Chore Core Protocol are as follows:

DEFINITION 4. ***(Separated Chore Core Protocol Properties)***

- *It does not change the main trim of pieces to a position on the right side.*
- *It does not change the main trim of at least one of the pieces.*

Algorithm 3: Existential Core Protocol

Data: Agent set $A = \langle a_1, a_2, \ldots, a_n \rangle$, specified cutter $a_{cutter} \in A$, and unallocated chore R

1 Specified cutter a_{cutter} divides the chore into n equal pieces according to her own perspective;
2 Define $p_1, p_2, \ldots, p_n$ the pieces that we have after the division of a_{cutter};
3 Define main trims $t_1, t_2, \ldots, t_n$ for pieces $p_1, p_2, \ldots, p_n$ respectively where they are initially on the rightmost side of the pieces;
4 Run Separated Chore Core Protocol on all the agents and all the pieces with their main trims. The call gives an envy-free partial allocation (at least the main trim of one of the pieces is not changed);
5 **return** envy-free partial allocation (at least one of the pieces has been completely allocated) and the unallocated chore;

In Algorithm 3, when we call Separated Chore Core Protocol, all the main trims are on the rightmost side of the pieces, but we make many other calls on Separated Chore Core Protocol such that the main trims are not necessarily on the rightmost side of the pieces. Separated Chore Core Protocol guarantees that its returned allocation does not change the main trim of at least one piece. Therefore, we can imply that from the cutter's perspective, at least $1/n$ of the chore is allocated. Existential Core Protocol, Algorithm 3, returns the allocation that Separated Chore Core Protocol returned. In the following Lemma we prove that if Separated Chore Core Protocol works, Existential Core Protocol works as well.

LEMMA 3.1. *If Separated Chore Core Protocol, Algorithm 4, works, Existential Core Protocol, Algorithm 3, gives an envy-free partial allocation to n agents in which one of the agents is the cutter who cuts the chore into n pieces, each agent gets a part of one of the pieces, and at least one agent gets a complete piece.*

Proof. If Algorithm 4 works correctly, its returned allocation is an envy-free partial allocation, and it does not change the main trim of at least one of the pieces. Since in Algorithm 3, we call Separated Chore Core Protocol for the pieces with a main trim on the rightmost side, at least one of the pieces is completely allocated in the returned allocation by Separated Chore Core Protocol.

3.2 Separated Chore Core Protocol In this Subsection we describe Separated Chore Core Protocol, Algorithm 4. As we mentioned in Subsection 3.1, this protocol receives a chore with n pieces as well as a set of n agents, and it returns an envy-free partial allocation of the pieces to the agents such that the main trim of at least one of the pieces remains intact. We say a piece is *intact* during a protocol P if its main trim does not change during the call of P.

This protocol is based on an iterative idea in lines 2-15 of Algorithm 4. After the i^{th} iteration of the loop we ensure that we have a neat allocation for the first i agents. We define a *neat* property for allocations as follows:

DEFINITION 5. *We call an allocation of m disjoint pieces of the chore to n agents (where $n \leq m$)* ***neat*** *if the following properties hold:*

- *The allocation allocates a (not necessarily whole) part of exactly one of the pieces to each agent.*
- *no agent prefers an unallocated piece or another agent's allocation to her allocation.*

In the i^{th} step of the loop, before running Line 15, we already have a neat allocation of pieces to the first i agents (we describe it in details later), and in Line 15, by running Best Piece Equalizer Protocol, it modifies the neat allocation. Best Piece Equalizer Protocol, Algorithm 7, is a protocol receiving a neat allocation of some pieces to some agents (one piece each agent), and it returns a modified neat allocation of pieces to the agents (one piece each agent). The properties of Best Piece Equalizer Protocol are as follows:

DEFINITION 6. *(**Best Piece Equalizer Protocol Properties**)*

- *The protocol is monotone.*
- *The returned allocation does not have any subset of bad agents.*

We define the monotonicity of a protocol as follows:

DEFINITION 7. *Assume that P is a protocol which receives a neat allocation of pieces to agent set A as input, and outputs another neat allocation of the pieces to the same set of agents. We call protocol P* ***monotone*** *if and only if it does not change the main trim of any piece p to the left side position.*

We also define a bad subset of agents as follows:

DEFINITION 8. *When we have a neat allocation of the pieces to some of the agents, we call a subset of agents S* ***bad*** *if the following conditions hold:*

- *One piece is allocated to each agent in S.*
- *None of the allocated pieces to the agents in S is intact.*
- *For each agent $a \in S$, the cost of the piece that we have allocated to a is less than the cost of any other piece.*

We describe Best Piece Equalizer Protocol in more details in Subsection 3.5.

Now, we describe how the protocol makes a neat allocation of pieces to the first i agents before running Best Piece Equalizer Protocol in Line 15. In the i^{th} step of the loop, agent a_i chooses piece p which is her most preferred piece among all pieces. However, p may be allocated before. If in the end of the $(i-1)^{th}$ step of the loop, p is not allocated to any agent, then we can simply allocate it to a_i (As mentioned before, we emphasize that when we allocate a piece to an agent, we allocate a partial part of it from its leftmost side to its main trim). Although we easily handled the case that p is not allocated, the other case is much harder. If p has been allocated to another agent a_j before, we have a conflict of interest on piece p. We define a *popular* piece and its *happy* or *sad fan* agents as follows:

DEFINITION 9. *When at least two agents a and b prefer a specific piece p to all other pieces, we call p a **popular piece**, and we call agents a and b the **fans** of piece p. We also call agent a a **happy fan** of p if she is a fan of p and p is already allocated to her, and we call her a **sad fan** of p if she is a fan of p but p is not already allocated to her.*

According to Definition 9, piece p is a popular piece, agent a_j is its happy fan, and agent a_i is its sad fan. We handle this conflict of interest based on two different cases whether the main trim of p is the same its initial main trim or not. First, we deal with the case that the main trim of p is not changed. In this case, we run Allocation Extender Protocol for all the pieces with their main trims, the first i agents with their current allocation, the popular piece p, and its fan agents a_i and a_j. Allocation Extender Protocol is a protocol which receives a neat allocation of pieces to the agents, a popular piece p, and its two specific happy and sad fan agents a and b. Note that in the allocation that this protocol receives, agent b is the only agent who does not have any piece. This protocol returns a neat allocation of pieces to the agents such that every agent has a piece and the main trim of piece p is not changed during the call of the protocol. We describe Allocation Extender Protocol in more details in Subsection 3.3.

Now, we deal with the case that p is not an intact piece. The general idea to handle this case is that to modify the current allocation such that an intact piece becomes the popular piece. Then, we can handle it similar to the previous case. We do this modification by calling Core Match Refiner Protocol, Algorithm 9. Core Match Refiner Protocol is a protocol which receives a neat allocation of pieces to agents, with a specific popular piece p. This protocol returns a new neat allocation and a flag variable with the following properties:

- In the new allocation piece p does not have any owner agent.
- If the flag is false, the new allocation has assigned a piece to each called agent.
- If the flag is true, the new allocation has assigned a piece to each called agent except one of them, who is the sad fan of one of the intact pieces.

We call Core Match Refiner Protocol for all the pieces with their main trims, the first $i-1$ agents with their current allocation, and the popular piece p. The call returns a refined neat allocation and a flag (In the case the flag is true, it returns the new popular piece q with agent a as its happy fan and agent b as its sad fan.) In the new allocation, piece p does not have any owner agent, so we can easily allocate it to a_i. If the flag is false, we have already increased the size of our neat allocation. If the flag is true, the situation is similar to the previous case such that there exists a popular intact piece q. Similar to the first case, we can run Allocation Extender Protocol (Line 14 of the Algorithm 4).

In the following lemma, we prove the correctness of Separated Chore Core Protocol.

LEMMA 3.2. *If Allocation Extender Protocol, Core Match Refiner Protocol, and Best Piece Equalizer Protocol work, Separated Chore Core Protocol, Algorithm 4, gives an envy-free partial allocation to the called agents such that Separated Chore Core Protocol Properties hold.*

3.3 Allocation Extender Protocol In this subsection, we describe Allocation Extender Protocol, Algorithm 5. This protocol receives a neat allocation of the pieces to the agents such that all the called agents has a piece except agent b who is the sad fan of the popular piece p. The goal of this protocol is to find a neat allocation which allocates a piece to each called agent and does not change the main trim of p.

In this protocol, first we ask agents a and b to trim each piece equal to p. For each piece q, we change its main trim to the rightmost trim made by a and b. Then, we deallocate all the allocated pieces but we remember the allocation as the *original mapping* and the main trim of the pieces as the *original mapping trims*. Then we run SubCore Protocol, Algorithm 6. SubCore Protocol is a protocol which receives a set of agents and pieces with an original mapping such that the main trim of each piece is not on the right side of its original mapping trim. It returns a neat allocation

Algorithm 4: Separated Chore Core Protocol

Data: A chore with n pieces $\langle p_1, p_2, \ldots, p_n \rangle$ with their main trims $\langle t_1, t_2, \ldots, t_n \rangle$ consecutively and a set of agents $A = \{a_1, a_2, \ldots, a_n\}$

Remember the initial main trims of the pieces during this call;
for $i = 1$ *to* n **do**
  Agent a_i chooses piece $p \in \{p_1, \ldots, p_n\}$ which is the most preferred piece for her among all pieces;
  if *p is not allocated to agents $a_1, a_2, \ldots, a_{i-1}$* **then**
    Allocate p to agent a_i;
  else
    Suppose that a_i chooses a piece which has been allocated to a_j;
    if *the main trim of p is not changed* **then**
      Run Allocation Extender Protocol for all the pieces with their main trims, the first i agents with their current allocation, the popular piece p, and its fan agents a_i and a_j. The call returns a neat allocation of pieces to agents (one piece each agent) without changing the main trim of p;
    else
      Run Core Match Refiner Protocol for all the pieces with their main trims, the first $i-1$ agents with their current allocation, and the popular piece p. The call returns a refined neat allocation and a flag (In the case the flag is true, it returns the new popular piece q with agent a as its happy fan and agent b as its sad fan);
      Allocate p to a_i;
      if *flag = true* **then**
        Run Allocation Extender Protocol for all the pieces with their main trims, the first i agents with their current allocation, the popular piece q, and its fan agents a and b. The call returns a neat allocation of pieces to agents (one piece each agent) without changing the main trim of q;
  Run Best Piece Equalizer Protocol on all n pieces with their main trims, all the first i agents, and the current allocation. The call gives a neat allocation of pieces to the called agents without any bad subset of agents;
return envy-free partial allocation (with at least one intact piece) and the unallocated chore;

Algorithm 5: Allocation Extender Protocol

Data: A chore with m pieces $\langle p_1, p_2, \ldots, p_m \rangle$ with their main trims $\langle t_1, t_2, \ldots, t_m \rangle$ consecutively, a set of agents $A = \{a_1, a_2, \ldots, a_n\}$ $(n \leq m)$ with a neat allocation of pieces to agents, one specific popular piece $p \in \{p_1, p_2, \ldots, p_m\}$, and its two specific happy fan agent $a \in A$ as well as sad fan agent $b \in A$

Remember the initial main trims of the pieces during this call;
for *each piece $q \in \{p_1, \ldots, p_m\}$* **do**
  Ask agents a and b to trim q equal to p;
  Set the main trim of piece q as the rightmost trim among the trims that agents a and b made on q;
Deallocate the allocated pieces but remember the allocation as the original mapping;
Run the SubCore Protocol on all m pieces except p with their main trims and all agents except a and b with their original mapping. The call gives a neat allocation of pieces to the called agents;
After the allocation, at least one piece q among pieces $\{p_1, \ldots, p_m\}$ except p is not allocated;
if *the main trim of q is made by agent a* **then**
  Allocate q to a, and p to b;
else
  Allocate q to b, and p to a;
return neat allocation of pieces to the agents (one piece each agent), without changing the main trim of p, and the unallocated chore;

of pieces to agents (one piece each agent) such that the following properties hold:

DEFINITION 10. ***(SubCore Protocol Properties)***

- *It does not change the main trim of unallocated pieces.*
- *It does not change the main trim of any allocated piece to a left side position.*
- *It does not change the main trim of a piece to a right side position of its original mapping trim.*

We describe this protocol in more details in Subsection 3.4. We emphasize that we do not change the original mapping during each call of SubCore Protocol. We call SubCore Protocol for all the first i agents except a and b with all the pieces except piece p. This call gives us a neat allocation of the called pieces to the called agents. After this call, we allocate a piece to a and another piece to b from the unallocated pieces in the following manner. We should have at least two unallocated pieces such that one of them is p. We take one of the other unallocated pieces q. First, we allocate q to the agent among a and b who made the main trim on it, and then, we allocate p to the other agent. Now, we have allocated a piece to each agent. In the following lemma, we prove that Allocation Extender Protocol works correctly.

LEMMA 3.3. *If SubCore Protocol works, Allocation Extender Protocol returns a neat allocation of pieces to agents (one piece each agent) such that p is an intact piece in this protocol.*

Proof. In the beginning of the Algorithm, agents a and b make trims on each piece equal to p. They can make this trim because p is their most preferred piece. Then, by calling SubCore Protocol, we have a neat allocation of pieces to all agents except a and b. Then, as we mentioned, we allocate an unallocated piece to each agent a and b. Without loss of generality, assume that agent a receives p and agent b receives q. Since SubCore Protocol returns a neat allocation to the called agents, They do not envy each other. They also do not envy to a, b or any unallocated piece, because after running SubCore Protocol, we have not changed the main trim of pieces. agents a and b do not envy the

other agents, because according to SubCore Protocol Properties, SubCore Protocol does not change the main trim of allocated pieces to a left side position, and it does not change the main trim of unallocated pieces. Agents a and b do not envy to an unallocated piece, because SubCore Protocol does not change the main trim of unallocated pieces. It is also easy to check that they do not envy each other.

3.4 SubCore Protocol In this Subsection, we describe Algorithm 6 in more details. As we mentioned before, this protocol gets a subset of agents and a subset of pieces with their main trims as well as the original mapping of agents to pieces. As we mentioned, in the original mapping, we have an allocation of pieces to agents (one piece each agent, and some of the pieces may be unallocated) such that in each call of the protocol, for each allocated piece in the original mapping, its original mapping trim is on the right side of its main trim. The protocol returns a neat allocation of pieces to the agents such that SubCore Protocol Properties, definition 10, hold.

In this protocol, similar to the idea that we used in Algorithm 4, we find a neat allocation iteratively. In Algorithm 6, we implement this loop in lines 2-26. We add the agents one by one, and in the end of the i-th step of the loop, we have a neat allocation of pieces to the first i agents. Similar to Algorithm 4 in the i-th step, we ask agent a_i to choose her most preferred piece. Assume that p is her choice, and it is not allocated to other agents. In this case, we can easily allocate p to a_i. Otherwise, we should handle the case in a much more complicated manner.

When agent a_i chooses a piece which is allocated to another agent, we have i agents that the first preference of each of them is among $i-1$ pieces. We call this set of allocated pieces P. We have $|P| = i-1$ allocated pieces, and we call these $i-1$ pieces *contested pieces*. The intuition to resolve this issue is to find a way to allocate one of the pieces outside of P to one of the first i agents. However, none of the first i agents may prefer these pieces. Hence, for using this idea, we need to change the main trim of some contested pieces and reallocate them to agents. To this end, we ask each of the first i agents to make a trim on each of the contested pieces to equalize it with the cost value of her most preferred piece outside of P. For an agent, if the cost value of a contested piece from its leftmost side to its original mapping trim was less than the cost value of her most preferred piece outside of P, She makes a trim on the original mapping trim of the contested piece. We define a representative agent for each piece in P (we have not assigned them yet.) If the trim of agent $a_j \in \{a_1, a_2, \ldots, a_i\}$ on contested piece q was the rightmost trim, and q was the original mapping of a_j, we set a_j as the representative of q. Otherwise, the agent with the rightmost trim and lowest index is its representative. We define set W as the set of agents who are the representative of at least one piece in P. In Lemma 3.4, we prove that W is the set of agents who are guaranteed to have a neat allocation. Therefore, our idea is to enlarge $|W|$ up to $|P|$ and make sure we have a neat allocation of P to agents in W. In each step of the loop in lines 15-24 of Algorithm 6, we increase the size of W by adding exactly one agent to it. In each step of this loop, we update the main trim of each piece in P by agents in $W' = \{a_1, \ldots, a_i\} \backslash W$ as follows: for each piece $p \in P$, we find the rightmost trim among the trims that the agents in W' have made on p, and update the main trim of p to this trim. Then, we find another mapping of pieces to agents in W' (one piece each agent), and we call it *modified original mapping*. We will use this mapping as the original mapping in our recursive call of SubCore Protocol. We find the modified original mapping as follows: we know that each agent in W is a representative of at least one piece in P. For each agent $a_j \in W$, if $p_{a_j} \in P$ and a_j was the representative of p_{a_j}, we assign p_{a_j} to a_j in the modified original mapping. Otherwise, we assign an arbitrary piece, from the set of pieces that a_j is their representative, to her. In both cases, the *modified original mapping trim* is the trim of a_j on the piece. The *modified original mapping trim* of a piece is its main trim in the modified original mapping. Moreover, the modified original mapping trim of each unallocated piece of P in the modified original mapping is the trim of its representative agent on it. In Lemma 3.4, we prove that this is an envy-free mapping of pieces to agents.

After finding the modified original mapping of W to P, we call SubCore Protocol on all pieces in P with their main trims, the agents in W, and the modified original mapping of W recursively. This call allocates a piece of P to each agent in W in a neat manner. If $|W| = |P|$, we break the loop, and among the first i agents, we have exactly one agent that we have not allocated any piece to her. Let a be this specific agent. We allocate the most preferred piece out of P for a to her. In Lemma 3.4, we prove that this allocation is feasible. Now, as the other case, assume that $|W| \neq |P|$. In this situation, we should further continue enlarging W. We take an arbitrary piece q from unallocated pieces of P. Let agent $a \in \{a_1, \ldots, a_i\} \setminus W$ be the agent who has the rightmost trim on q (In Lemma 3.4, we prove that this trim is the main trim of q.) If more than one such agent exists, a is the agent with the lowest index. We allocate q to a and add a to W. Note that in the beginning

Algorithm 6: SubCore Protocol

Data: A chore with m pieces $\langle p_1, p_2, \ldots, p_m \rangle$ with their main trims $\langle t_1, t_2, \ldots, t_m \rangle$ consecutively and a set of agents $A = \{a_1, a_2, \ldots, a_n\}$ with an original mapping of agents to pieces $\{p_{a_1}, p_{a_2}, \ldots, p_{a_n}\}$ consecutively

1 Define a set P of pieces which is an empty set initially;
2 **for** $i = 1$ *to* n **do**
3 Agent a_i chooses piece p which is the most preferred piece for her among the pieces;
4 **if** *p has not been allocated to agents* $a_1, a_2, \ldots, a_{i-1}$ **then**
5 Allocate p to agent a_i tentatively and add p to set P;
6 **else**
7 Define a representative agent for each piece in P (not assigned yet);
8 **for** $j = 1$ *to* i **do**
9 Find the cost value of each piece outside of P for a_j. We define c_j as the minimum cost value among these values;
10 **for** *each piece* $q \in P$ **do**
11 a_j makes a trim on q in a way that equalize it to c_j. If the cost value of the piece from its leftmost side to its original mapping trim was not more than c_j, a_j makes a trim on the original mapping trim of the piece;
12 **if** *a_j makes the rightmost trim on q* **then**
13 Set a_j as the representative of q if she makes the first rightmost trim on it, or $q = p_{a_j}$ holds;
14 Define set W which initially contains the representative agents of the pieces in P;
15 **while** $|W| \leq |P|$ **do**
16 Ignore the previous trims of agents in W and deallocate the allocated pieces;
17 Define set $W' = \{a_1, \ldots, a_i\} \setminus W$;
18 Update the main trim of pieces in P by agents in W' (check Subsection 3.4 for more details);
19 Find the modified original mapping of W to P (check Subsection 3.4 for more details);
20 Run SubCore Protocol on all pieces in P with their main trims, the agents in W, and the modified original mapping of W as the original mapping. The call gives an allocation of pieces to agents in W;
21 **if** $|W| = |P|$ **then**
22 Break;
23 Take an arbitrary piece q from unallocated pieces of P such that $a \in W'$ is the agent with the lowest index among the agents who have the rightmost trim on q;
24 Allocate q to a and add a to W;
25 Let a be the only agent among $a_1, \ldots, a_i$ who is not in W;
26 Find the cost value of each piece out of P for a from its leftmost side to its main trim, and allocate the piece with the minimum cost value to a;
27 **return** an allocation of pieces to A (one piece each agent) with their updated main trims;

of this loop we ignore the previous trims of agents in W and deallocate the allocated pieces, because we will find another allocation by calling SubCore Protocol recursively.

In the following lemma, we prove that our SubCore Protocol guarantees a neat allocation, and all the SubCore Protocol Properties hold.

LEMMA 3.4. *Suppose that we have set A of n agents and m pieces where $n \leq m$. Each of the pieces has an original mapping trim as well as a main trim such that the main trim of each piece is not on the right side of its original mapping trim. Also, assume that there is a neat allocation for the agents and the pieces with their original mapping (using their original mapping trims). Then, calling the SubCore Protocol for these agents and pieces returns a neat allocation of pieces to agents such that all the SubCore Protocol Properties that we mentioned in Definition 10 hold.*

3.5 Best Piece Equalizer Protocol In this Subsection, we describe Best Piece Equalizer Protocol. Assume that we have a neat allocation of pieces to some agents. The goal of Best Piece Equalizer Protocol is that to update the neat allocation and the main trims of the allocated pieces such that all the Best Piece Equalizer Protocol Properties, Definition 6, hold. This protocol is based on a loop in Lines 1-12. We call it as the *main loop* of the protocol. In each step of the main loop we find a bad subset of agents, and by changing the current allocation, we make it a non-bad subset. This loop ends when we do not have anymore bad subset. Assume that S is a bad subset of agents in a step of the main loop, and P is the set of pieces that we have allocated to S. While S is a bad set, we ask each agent $a \in S$ to makes a trim on each piece $p \in P$ to equalize it with her most preferred piece out of P (If the cost value of p from its leftmost side to its rightmost side was less than the cost value of her most preferred piece out of P, she makes a trim on the rightmost side of p.) We call the leftmost trim on p which is on the right side of its main trim as the *equalizer trim* of p, and we call the current allocation of pieces to agents as the *old allocation*. We run Separated Chore Core Protocol on all the agents in S and all the pieces in P with their equalizer trims as their main trims. The call returns an envy-free partial allocation of pieces to agents such that all Separated Chore Core Protocol Properties that we mentioned in Definition 4 hold. We call the returned allocation the *new allocation*. In the new allocation, the main trim of a piece may change to the left side of its initial main trim. This makes our protocol non-monotone. We guarantee the monotonicity of the protocol by running Monotonicity Saver Protocol. Monotonicity Saver Protocol is a protocol receiving n agents as well as n pieces, a neat allocation of pieces to agents (one piece each agent) as the *old allocation*, and a neat allocation of pieces to agents (one piece each agent) as the *new allocation*. This protocol returns another neat allocation such that the main trim of each piece in the result allocation is not on the left side of the main trim of the

Algorithm 7: Best Piece Equalizer Protocol

Data: A chore with m pieces $\langle p_1, p_2, \ldots, p_m \rangle$ with their main trims $\langle t_1, t_2, \ldots, t_m \rangle$ consecutively and a set of n agents $A = \{a_1, a_2, \ldots, a_n\}$ $(n \leq m)$ with an allocation of the pieces $\{p_{a_1}, p_{a_2}, \ldots, p_{a_n}\}$ to the agents $\{a_1, a_2, \ldots, a_n\}$ consecutively

while *there exists a bad subset* $S \subseteq A$ *of the agents* **do**
  Define set P as the set of pieces that we have allocated to S;
  while S *is a bad set* **do**
    for *each agent* $a \in S$ **do**
      for *each piece* $p \in P$ **do**
        Ask agent a to make a trim on p to equalize it with her most preferred piece out of P (If the cost value of p from its leftmost side to its rightmost side was less than the cost value of her most preferred piece out of P, she makes a trim on the rightmost side of p.);
    Define an equalizer trim for each piece in P (We have not set them yet);
    for *each piece* $p \in P$ **do**
      Set the leftmost trim on p which is on the right side of its main trim as the equalizer trim of p;
    Set the current allocation of pieces to agents as the old allocation;
    Run Separated Chore Core Protocol on all the agents in S and all the pieces in P with their equalizer trims as their main trims. The call gives an envy-free partial allocation of the pieces to agents, which we call it the new allocation, and updates the main trims (At least the main trim of one of the pieces is not changed);
    Run Monotonicity Saver Protocol for the agents in S, the pieces in P, and the old as well as new allocations of pieces to S. The call gives an envy-free partial allocation of the called pieces to the called agents, and it updates the main trims to keep the protocol monotone.
return a neat allocation (without any bad subset of agents) of pieces to A (one piece each agent) with their updated main trims;

secondary allocation. We run Monotonicity Saver Protocol for the agents in S, the pieces in P, and the old as well as the new allocation. We update our current allocation with the returned allocation by Monotonicity Saver Protocol.

In the following lemma, we prove the correctness of Best Piece Equalizer Protocol if Separated Chore Core Protocol and Monotonicity Saver Protocol work correctly.

LEMMA 3.5. *If Separated Chore Core Protocol and Monotonicity Saver Protocol work correctly, Best Piece Equalizer Protocol returns a neat allocation (one piece each agent), and all the Best Piece Equalizer Protocol Properties, that we mentioned in Definition 6, hold.*

Proof. First of all, we prove that in each iteration of the main loop we make subset S of the agents a non-bad subset. When we ask each agent to make a trim on each piece p in P to equalize it with her most preferred piece out of P, at least the trim of the owner of p is on the right side of its main trim, because S is a bad subset of agents, and according to the definition, the cost value of each piece for its owner in P should be less than her most preferred piece out of P. Thus, we can guarantee that the two following properties:

- **Property 1**: The equalizer trim of each piece exists.
- **Property 2**: Each agent in S has a trim on at least one of the pieces which is not on the left side of the main trim of that piece.

Then, by running Separated Chore Core Protocol, according to its properties in Definition 4, we receive an allocation such that at least one of the called pieces is intact, and the main trim of any other piece is not changed to a position on the right side. Before running Separated Chore Core Protocol, we have an allocation, and we call it the old allocation. After running Separated Chore Core Protocol we have another allocation, and we call it the new allocation. Now, we prove that the allocation that we have after running Montonicity Saver Protocol is neat, but before that let S_1 be the subset of S such that the main trim of their pieces have not changed by the Monotonicity Saver Protocol, and $S_2 = S \setminus S_1$.

- proof of envy-freeness in S: Since the returned allocation by Monotonicity Saver Protocol is envy-free partial, the agents in S do not envy each other.
- proof of envy-freeness of agents in S to agents out of S and unallocated pieces: As we mentioned, Separated Chore Core Protocol returns an allocation such that none of the main trims has changed to a position on the right side. Thus, according to Property 2, we can infer that in the allocation returned by Separated Chore Core Protocol for each agent a the cost value of at least one of the pieces in P is not more than the cost value of her most preferred piece out of P. Based on this and envy-freeness of the returned allocation by Separated Chore Core Protocol, we can infer that after running Separated Chore Core Protocol and before running Monotonicity Saver Protocol, the agents in S do not prefer any piece out of P to their allocated pieces. Since after running Monotonicity Saver Protocol the allocated pieces to S_1 (and their main trims) have not changed, the agents in S_1 do not envy to agents out of S and unallocated pieces. Moreover, in the Monotonicity Saver Protocol, we have changed the allocated pieces to S_2 to their allocated pieces in the old allocation. In the old

allocation, they did not prefer the pieces out of P to their own piece. Therefore, they do not prefer these pieces again.

- proof of envy-freeness of agents out of S to agents in S: the main trim of allocated pieces to agents in S_1 have not changed to a left side position, and the main trim of pieces allocated to agents in S_2 have not changed. Therefore, since before running Separated Chore Core Protocol the agents out of S did not envy the agents in S, after running Monotonicity Saver Protocol, we keep this envy-freeness property.

By running Monotonicity Saver Protocol, we make sure that after each iteration of the main loop, the main trim of any piece has not changed to a left side position. Therefore, we can infer that our protocol is monotone.

Unfortunately, the main loop of our protocol may iterate infinite times, but the good property of our protocol is that in each iteration, it changes the main trim of some pieces (at least one piece) to a position on the right side, and it does not change the main trim of any piece to a left side position. In the main loop we change the main trim of at least one of the pieces to a position on the right side. The reason is that all the equalizer trims are on the right side of the main trims, and by running Separated Chore Core Protocol at least one of the called pieces remains intact. Since each piece has a rightmost side, a limit, and our protocol is monotone, and it changes the main trim of at least one piece in each iteration, we can infer that we converge to a neat allocation solution if the protocol does not finish in finite iterations.

3.6 Monotonicity Saver Protocol In this subsection, we describe Monotonicity Saver Protocol, Algorithm 8. As we mentioned in Subsection 3.5, this protocol receives a set of n pieces, a set of n agents, and two different allocation of pieces to agents with different main trims. We call these allocations the new and the allocations. Our goal is to change the new allocation such that the main trim of each piece does not be on the left side of its main trim in the old allocation.

Let P be the set of pieces whose main trim in the new allocation is on the left side of its main trim in the old allocation, and let S be the set of agents who own the pieces in P. First, we deallocate the pieces that we have allocated to the agents of S in the new allocation. Then, for each agent $a_i \in S$, we change the main trim of p_{a_i} to its main trim in the old allocation, and we allocate it to a_i. In Lemma 3.6, we prove that this allocation is feasible and the protocol works correctly.

Algorithm 8: Monotonicity Saver Protocol

Data: A chore with n pieces $\langle p_1, p_2, \ldots, p_n \rangle$, a set of n agents $A = \{a_1, a_2, \ldots, a_n\}$, an old allocation of the pieces $\{p_{a_1}, p_{a_2}, \ldots, p_{a_n}\}$ to the agents $\{a_1, a_2, \ldots, a_n\}$ consecutively, a new allocation of pieces to agents, and the main trims of the allocations

Let P be the set of pieces whose the main trim in the new allocation is on the left side its main trim in the old allocation;
Let S be the set of agents who owns the pieces in P;
Deallocate the pieces that we have allocated to S in the new allocation;
for *each agent* $a_i \in S$ **do**
    Change the main trim of p_{a_i} to its main trim in the old allocation;
    Allocate p_{a_i} to a_i;
return the updated new allocation which is an envy-free partial allocation of pieces to A (one piece each agent);

Algorithm 9: Core Match Refiner Protocol

Data: A chore with m pieces $\langle p_1, p_2, \ldots, p_m \rangle$, a set of n agents $A = \{a_1, a_2, \ldots, a_n\}$ $(n \leq m)$ with a neat allocation of pieces to agents (without any bad subset of agents), and a specific popular piece $p \in \{p_1, p_2, \ldots, p_m\}$

Build the Core Match Refiner Graph G;
Define a directed path P which is initially $P = \langle v_p \rangle$ where v_p is the vertex of piece p;
while *the piece of the last vertex of* P *is not an unallocated or an intact piece* **do**
    Find a vertex v_r out of P such that there exists an edge from v_q one of the vertices of P to it. Let v_r and v_q be the vertices of pieces r and q consecutively;
    Remove all the vertices after v_q from the path;
    Add v_r to the end of the path;
Let $P = \langle v_1, v_2, \ldots, v_k \rangle$ be the final path;
Define a flag which is false if the piece of vertex v_k be an unallocated piece, and true otherwise;
if *flag = true* **then**
    Let agents a and b be the owners of the pieces of vertices v_{k-1} and v_k consecutively;
    Let piece q be the piece of vertex v_k. Define q as a popular piece with agents a and b as its fans;
    Deallocate the piece of vertex v_k from agent b;
for i *from* $k-1$ *to* 1 **do**
    Deallocate the piece of vertex v_i from its owner agent c;
    Allocate the piece of vertex v_{i+1} to agent c;
return the refined neat allocation and the flag (In the case the flag is true, return the popular piece q, its happy fan a, and sad fan b) ;

LEMMA 3.6. *Monotonicity Saver Protocol returns an envy-free partial allocation of pieces to agents (one piece each agent) such that the main trim of each piece in the returned allocation is not on the left side of its main trim in the old allocation.*

3.7 Core Match Refiner Protocol In Core Match Refiner Protocol, Algorithm 9, we build a special graph that we call it the *Core Match Refiner Graph*. We build Core Match Refiner Graph G as follows: For each piece we have a vertex in G. We connect the vertex of piece p to the vertex of piece q with a directed edge if and only

if $C_a(p) = C_a(q)$ where agent a is the owner of p.

In the following lemma, we prove that Core Match Refiner Protocol works correctly.

LEMMA 3.7. *Core Match Refiner Protocol works correctly.*

3.8 Final Remarks In the previous subsections, for each protocol P we prove that if each of the protocols that P calls it works correctly, P works correctly as well. In Figure 1, we have a graph of these protocols. Protocol P_1 has an edge to Protocol P_2, if P_1 calls P_2. If the graph did not have a loop, we could infer that the whole procedure works correctly. However, we have two loops in the graph. The first loop happens in the SubCore Protocol such that it calls itself recursively. Anytime this protocol calls itself, the call happens on a smaller subset of agents, and we know that when the number of agents is one, it does not call itself anymore. The second loop happens between Separated Chore Core Protocol and Best Piece Equalizer Protocol. Similarly, each time Best Piece Equalizer Protocol calls Separated Chore Core Protocol, the call happens on a smaller number of agents. Moreover, Separated Chore Core Protocol does not call Best Piece Equalizer Protocol on a larger number of agents. We also know that when the number of agents is one, Separated Chore Core Protocol does not call Best Piece Equalizer Protocol anymore. Therefore, based on a simple induction we can infer that none of these loops makes any issue in the correctness of the whole procedure.

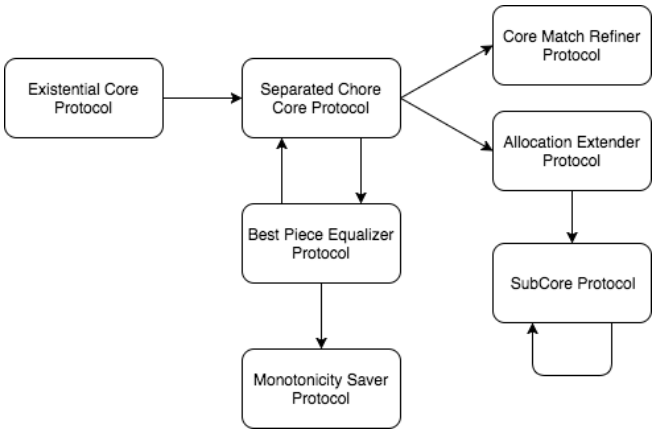

Figure 1: The Graph of the protocols in the Core. If Protocol P_1 has an edge to Protocol P_2, it means that P_1 calls P_2.

Running Existential Core Protocol returns an envy-free partial allocation. As we mentioned in Subsection 3.5, we may need infinite number of queries in each call of Best Piece Equalizer Protocol. This makes Existential Core Protocol unbounded, but since Best Piece Equalizer Protocol is a monotone protocol with an upper-bound, as we discussed in Subsection 3.5, it converges to a solution.

Now, in the following lemma using Existential Core Protocol, we guarantee that Core Protocol, Algorithm 2, returns an envy-free partial allocation.

LEMMA 3.8. *Core Protocol, Algorithm 2, is a discrete and bounded algorithm that returns an envy-free partial allocation of pieces to agents (one piece each agent) such that at least one of the pieces is completely allocated.*

Proof. The algorithm is clearly discrete and bounded. Now, we show that there exists an ordering of agents and pieces which guarantees an envy-free partial allocation. It suffices to show that the returned allocation of Existential Core Protocol can be made by one of the orderings of agents and pieces in Core Protocol. To this end, we analyze the properties of the returned allocation by Existential Core Protocol. Existential Core Protocol guarantees that we have allocated a complete piece to at least one of the agents. Without loss of generality, assume that agent a is this agent, and piece p is allocated to her. We create two lists V_A and V_P of the agents and the pieces consecutively. They finally will be the ordering of the agents and the pieces that we are looking for. We add agent a to V_A and piece p to V_P. In the end of each iteration of the main loop of Separated Chore Core Protocol we call the best piece equalizer protocol. Therefore, in the returned allocation we should not have any bad subset of agents. First of all, for each agent $b \neq a$ who has received a complete piece q, we add b to V_A and q to V_P. Now, we iteratively add the other agents and pieces to V_A and V_P. Since there exists no bad subset of agents and none of the agents outside of V_A has received a complete piece, at least one of the agents outside of V_A should prefer one of the pieces in V_P as the same as her own piece. We add this agent to the end of V_A and its piece to the end of V_P. We do this iteratively until we do not have anymore agent outside of V_A. Now, we claim that the ordering of V_A and V_P guarantees an envy-free allocation in Core Protocol, Algorithm 2. Since the returned allocation of Existential Core Protocol is envy-free, the allocation is envy-free. Now, for each piece, we should make sure the trim that an agent makes on it in Core Protocol, Algorithm 2, in the ordering of V_A and V_P is equal to its main trim in the returned allocation by Existential Core Protocol. The completely allocated pieces in the allocation of Existential Core Protocol are the first pieces in V_P. In Core Protocol, in the ordering of V_A and V_P, the agents trim the rightmost side of these pieces similar to Existential Core Protocol. For

the other pieces, in Core Protocol, the agents trim the pieces to equalize them to their most preferred piece among the previously allocated pieces. In the ordering of V_A and V_P, in the returned allocation by Existential Core Protocol, we also know that for the incomplete pieces, their owners prefer at least one of the previously allocated pieces as much as their own piece. It implies that we have the same trims on the returned allocations of both protocols for the incomplete pieces as well.

Moreover, we should mention that the first agent in any ordering of agents in Algorithm 2 receives a complete piece. Thus, we have allocated a complete piece to at least one agent.

As we mentioned in Introduction, we can use a similar approach to simplify the Core Protocol of cake cutting. Algorithm 10 simplifies the Core Protocol proposed by Aziz and Mackenzie [3]. The idea of this algorithm is similar to Algorithm 2. We prove that there exists an ordering of the agents and an ordering of the pieces such that the following protocol provides an envy-free allocation. Agents receive their pieces one by one. The first agent receives the first piece. The i^{th} agent trims the i^{th} piece in such a way that she receives the smallest part of it without envying the first $i-1$ agents. In the following Theorem we prove Algorithm 10 returns a partial envy-free allocation for the Cake Cutting problem.

Algorithm 10: Core Protocol - Cake Cutting

Data: Agent set $A = \langle a_1, a_2, \ldots, a_n \rangle$, specified cutter $a_{cutter} \in A$, and unallocated cake R

Specified cutter a_{cutter} divides the cake into n equal pieces according to her own perspective;
Define $p_1, p_2, \ldots, p_n$ the pieces that we have after the division of a_{cutter};
for *each permutation* $\langle a'_1, a'_2, \ldots, a'_n \rangle$ *of the agents* **do**
  for *each permutation* $\langle p_{a'_1}, p_{a'_2}, \ldots, p_{a'_n} \rangle$ *of the pieces* **do**
    Allocate $p_{a'_1}$ to a'_1 completely;
    for *i from 2 to n* **do**
      Ask a'_i to trim $p_{a'_i}$ to equalize it with her most preferred piece among the first $i-1$ allocated pieces (we consider the value of each piece from its leftmost side to its trim.);
      Allocate $p_{a'_i}$, from its leftmost side to its trim, to a'_i;
    if *none of the agents envies to another agent* **then**
      return the envy-free partial allocation (at least one of the pieces has been completely allocated) and the unallocated cake;
    Ignore the previous trims and deallocate the allocated pieces;

THEOREM 3.1. *Algorithm 10, is a discrete and bounded algorithm that returns an envy-free partial allocation of pieces to agents (one piece each agent) such that at least one of the pieces is completely allocated.*

4 Permutation Protocol

In Permutation Protocol, we are given a large number of isomorphic snapshots. This protocol modifies these snapshots such that a set of agents $B \subset A$ get a significant advantage over other agents and returns B. Recall that each snapshot is a partial envy-free allocation returned by Core Protocol. Therefore, in each snapshot at least one agent has a very significant advantage over another agent. This means that at least one agent thinks that the cost of the piece which is allocated to her is significantly smaller than other agents'. Let a be the agent with a very significant advantage over another agent b. The main idea in this protocol is if we exchange allocated pieces such that another agent gets the piece which was allocated to b, then agent a gets a significant advantage over the agent who receives this piece. Exchanging pieces does not necessarily preserve envy-freeness. In order to maintain envy-freeness, the protocol modifies some snapshots every time it wants to exchange some pieces. Permutation Protocol modifies some allocated pieces by detaching small part of them and ensures that the cost of the detached pieces are very insignificant to every agent. Therefore, in the case an agent a has a very significant advantage over some agent b in a snapshot, if we modify agent b's piece and give it to some other agent c, then agent a gets a significant advantage over c. We declare *working set* to be the set of snapshots that protocol is working with them and use W to denote it. The protocol successively removes some snapshots from the working set and modifies them and then exchanges some pieces in the snapshots of the working set. Also, whenever we exchange some pieces, we reserve a snapshot, i.e., we remove one snapshot from our working set and do not modify this snapshot anymore. The reservation is crucial for our protocol, since an agent might lose her advantage over others if we exchange the allocated pieces.

Recall that in Main Protocol every agent is placed a trim on other agents' pieces to make it equal to her piece. Since the snapshots are isomorphic, the order of trims is the same in all the snapshots. Moreover, we always modify all the snapshots in the working set in the same way to preserve isomorphism.

For an agent a we use p_a to denote the piece which is allocated to a in all the snapshots and $t^a_1, t^a_2, \cdots, t^a_{l_a}$ to denote the trim lines from right to left on p_a where l_a is the number of trim lines on p_a and we use $d^a_1, d^a_2, \ldots, d^a_{l_a}$ to indicate the agents with a trim on p_a from right

Algorithm 11: Permutation Protocol

Data: List of agents $A = \{a_1, \ldots, a_n\}$ and IS_n isomorphic snapshots

Set boolean $done = false$;
Add all the snapshots to the working set W;
while $done = false$ **do**
  Create an empty graph G ;
  for *each active piece* p_a **do**
    Add a directed edge from a to the agent with the next rightmost trim on p_a ;
  for *each inactive piece* p_b **do**
    Add a directed edge from b to all other agents ;
  Let $C = b_1, b_2, \ldots$ be the cycle with at least one active node ;
  for *each* b *in* C **do**
    Let c be the next agent in the cycle ;
    if b *is an active node* **then**
      Relabel the agents in order of their trims on p_b, such that agent a_1 is the first agent who had this piece, agent a_2 detached a first part and so on. Suppose that the trim of agent a_k is already detached and agent a_k is the owner of this piece and we want to detach from the trim of next agent which is a_{k+1}. Note that $b = a_k$ and $c = a_{k+1}$;
      Create a new working set W';
      while $|W| \geq k + 1$ **do**
        Pick $k + 1$ snapshots $s_1, s_2, \cdots, s_{k+1}$ from W and remove them from W;
        Run Cake Subcore Protocol with the pieces $e_{k+1}^{s_1,a_k}, e_{k+1}^{s_2,a_k}, \cdots, e_{k+1}^{s_{k+1},a_k}$ and agents $a_1, a_2, \cdots, a_k$;
        Let s' be a snapshot whose piece is unallocated. Add s' to W';
        for $i = 1$ *to* $k + 1$ **do**
          if $s_i \neq s'$ **then**
            Let d be the agent that $e_{k+1}^{s_i,a_k}$ is allocated to her in Cake Subcore Protocol. According to observation 1 reallocate the pieces in s_i, such that d becomes the owner of ${s_i}^{a_k}$;
            Detach the part which is allocated to d in Cake Subcore Protocol.
        if *detached parts cost significant for at least one agent* **then**
          Let X be the union of detached parts ;
          return Discrepancy Protocol(X,R) ;
      $W = W'$;
  for *each* b *in* C **do**
    if b *is an active node* **then**
      Detach the next part in p_b in all the working set's snapshots.
    else
      Reattach the detached pieces of p_b until we reach the trim line of agent c ;
  Exchange the pieces of agents in C, such that each agent receives the next agent's pieces. ;
  while *there exists an active piece* p_a *such that all its trims are detached* **do**
    if p_a *had less than* $n - 1$ *trims* **then**
      Let B be the set of agents without a trim on this piece, and make done = $true$;
      return B ;
    else
      Deactivate p_a ;
  Reserve a snapshot and remove it from W;

to left. Moreover, we can partition p_a based on the trim lines. We use e_i^a to denote the part of p_a which is between two consecutive trims and the left trim is t_i^a. Recall that, Main Protocol ensures that for every e_i^a, the mask of this part is the same across all the snapshots, i.e., the agents who think the cost of this part is very significant are the same in all the snapshots.

Trim lines on the pieces are the guidelines for modifying the allocation. Consider an arbitrary piece p_a. If this piece has no trim on it, this means that all other agents have a significant advantage over this agent, so we are done. Otherwise, consider the rightmost trim of this piece which is t_1^a, if we cut p_a from this trim, then agent who has this trim is willing to exchange his piece for p_a. This is the basic idea behind Permutation Protocol. It successively detaches some part of allocated pieces to make it desirable for other agents and then exchanges the allocated pieces.

At each iteration, Permutation Protocol creates an empty graph G with n nodes. It adds a directed edge from node a to b if the next trim line on the agent a's piece is for b. We add directed edges from a to all other nodes in case that all the trim lines on the agent a's piece are already detached. Recall that an agent places a trim on other agent's piece if the piece she has is not significantly smaller than this piece. Therefore, if a piece has less than $n - 1$ trims and we detach all of them, the set of agents who did not have a trim on this piece get a significant advantage over all other agents since every other agent has this piece in some reserved snapshot, and the protocol can return this set of agents. Therefore, we can assume that every piece that its trim lines are detached had $n - 1$ trim lines. We call these pieces *inactive* and the others *active*.

DEFINITION 11. *We call a piece* ***inactive*** *if it has* $n-1$ *trims and all its trim lines are detached. We call every other piece an* ***active*** *piece. We call a node in the graph inactive if its corresponding piece is inactive and otherwise active.*

Considering an inactive piece, all the agents had this piece at some point. We can easily make this piece desirable for every other agent by reattaching some of the detached parts. Therefore, we can add directed edges from inactive pieces to all other nodes. Note that the Core Protocol gives a significant advantage to the cutter over some other agent, so at least one piece has less than $n-1$ trim lines. Similar to the approach used in [3] it can be shown that the graph always contains a cycle with at least one active node.

Let C be a cycle in graph G with at least one active node. We want to detach the next parts of corresponding pieces of active nodes. By detaching

some part of a piece and giving it to another agent, some other agents might envy. Specifically, if $e_1^a, \cdots, e_k^a$ are already detached from p_a and we detach e_{k+1}^a and give this piece to d_{k+1}^a, agents a and $d_1^a, \dots d_k^a$ might envy d_{k+1}^a since the cost of this piece is less than what they have. However, the amount that each of these agents envy the agent who receives this piece, is at most the cost of the e_{k+1}^a. To ensure that no envy arises during the exchanging the pieces, we must also detach part of other agents' pieces such that the cost of the detached piece for each of them be as large as e_{k+1}^a. Moreover, in order to prevent any enviousness between these agents, the cost of the part that we detach from each of them should be as large as the others'. The following lemma is our main tool to achieve this goal.

LEMMA 4.1. (CAKE SUBCORE PROTOCOL) *Given $k+1$ pieces $p_1, p_2, \cdots, p_{k+1}$ and k agents $a_1, a_2, \cdots, a_k$, there exists a bounded protocol which finds an allocation with the following properties.*

- *Each agent's allocation is a part of one piece.*
- *One piece is unallocated.*
- *No agent prefers the unallocated piece or an other agent's allocation over her own allocation.*
- *At least one piece is completely allocated to the agents.*

Suppose that we want to detach a part of piece p. We relabel the agents in order that they had a trim line on this piece, so that agent a_1 was the first agent who had this piece, agent a_2 detached a first part and so on. Assume that trim line of agent a_k is already detached and agent a_k is the owner of this piece and we want to detach the next part of this piece which is $e_{k+1}^{a_k}$ and give it to agent $d_{k+1}^{a_k}$ which is a_{k+1} since we have relabeled the agents. Let A' be the set of agents who had this piece which is $A' = \{a_1, \cdots, a_k\}$. If we detach $e_{k+1}^{a_k}$ and give p to agent a_{k+1}, agents in A' may envy this agent. Therefore, for every agent in A' detach some part of her allocated pieces to preserve envy-freeness. We pick $k+1$ snapshots from our working set. Let $s_1, s_2, \cdots, s_{k+1}$ be these snapshots. We call Cake Subcore Protocol (Lemma 4.1) with the pieces $e_{k+1}^{s_1,a_k}, e_{k+1}^{s_2,a_k}, \dots, e_{k+1}^{s_{k+1},a_k}$ and agents A'. It gives an allocation such that every agent thinks her allocated piece costs more than other agents' allocations and the unallocated piece. If we can detach these allocated parts from its owner, then every agent thinks that the cost of the part that we detach from her is as large as the others'. Let s be a snapshot whose part is unallocated in the allocation returned by Cake Subcore Protocol. Detaching the allocated parts, makes exchanging s agreeable for all the agents, since each agent thinks that the part detached from herself costs more than e_{k+1}^{s,a_k}. The only problem is that current owner of piece p in all snapshots is agent a_k. The following observation shows that we can reattach some parts and change the allocation while preserving envy-freeness such that each agent becomes the owner of the piece that she wants.

OBSERVATION 1. *Given snapshot s and piece p in this snapshot, let a be the agent whose trim is detached from p. We can reattach some of the detach pieces and change the allocation of the pieces such that a becomes the owner of p, and no envy arises.*

Suppose that we want to detach the next part of piece p. Assume that agent a_1 was the original owner of her piece, and trim of agents $a_2, a_3, \dots, a_k$ are detached, and we want to detach the trim line of the agent a_{k+1}. We take $k+1$ snapshots from the working set $s_1, s_2, \dots, s_{k+1}$, and call Cake SubCore Protocol with the pieces $e_{k+1}^{s_i,a_k}$ for every s_i, and agents $a_1, a_2, \cdots, a_k$. Let s be a snapshot whose part is not allocated to any agent in this call, and also assume that snapshot piece $e_{k+1}^{s_i,a_k}$ is allocated to agent a_j, by the observation 1, we edit the snapshot s_i such that a_j becomes the new owner of piece p, and then detach that is allocated to her in the Cake SubCore Protocol from her piece. This makes the exchanging of s agreeable for all the agents, and we add w to our new working set.

While we are detaching some parts, an agent might think that the cost of the detached parts is very significant. Recall that mask of every $e_{k+1}^{a_k}$ is the same, so by partially detaching some of the $e_{k+1}^{a_k}$ parts, at least one agent thinks that the cost of the detached part is very insignificant. Thus, we find a high discrepancy in agents' valuation of detached parts in comparison to R. We give this discrepant part to Discrepancy Protocol to allocate the whole chore.

After making all the exchanges agreeable for the agents, we detach the next part of every piece that we want to exchange, and then we exchange the pieces. The protocol successively does the same until it finds a piece that has less than $n-1$ trims and all its trims are detached. Therefore, it finds a set of agents with a significant bonus over others and returns this set of agents.

5 Discrepancy Protocol

Discrepancy Protocol is responsible for dominating a set of agents to others whenever we find an unallocated piece such that there is a high discrepancy in the agents' valuation of this piece.

Assume that we have detached a piece that is very

Algorithm 12: Discrepancy Protocol

Data: List of agents $A = \{a_1, \ldots, a_n\}$, a discrepant piece e and residual chore R

Let B be the set of agents who thinks p costs very significant. $B = \{\forall a_i : C_{a_i}(e) \geq C_{a_i}(R) \times 2^n\}$;
Let B be the other agents. $C = A \setminus B$;
Let P_C be the output of Near Exact(B,e,$2^{|C|+1}$,$1/(2^{|C|+2})$) ;
Let P_B be the output of Near Exact(C,R,$2^{|B|+1}$,$1/(2^{|B|+2})$) ;
Call Oblige Protocol(C,P_C) ;
Call Oblige Protocol(B,P_B) ;
Let R_B be the unallocated parts of P_B ;
Run Main Protocol(B,R_B) ;
Let R_C be the unallocated parts of P_C ;
Run Main Protocol(C,R_C) ;
return B ;
(Since the whole chore is already allocated, every agent dominates the others.)

Algorithm 13: Oblige Protocol

Data: List of agents $A = \{a_1, \ldots, a_n\}$ and a partition of the chore into 2^{n+1} pieces $p_1, \ldots, p_{2^{n+1}}$.

for $i = 1 to n$ **do**
  Ask agent a_i to set aside her 2^{i-1} largest pieces. ;
for $i = n to 1$ **do**
  Ask agent a_i to return part of her reserved pieces to the remaining pieces to create a $2^{i-1}+1$-way tie for her smallest pieces ;
for $i = 1 to n$ **do**
  Ask agent a_i to choose her smallest piece in the remaining pieces. Each agent have to take a piece she augmented if one is available;
  (Break ties in lexicographic order.)
return partial envy-free allocation and remaining of the chore ;

significant for a set of agents B and very insignificant for the others. Since the modifications in Permutation Protocol were very insignificant for all the agents, for each agent a_i in B we have:

$$C_{a_i}(e) \geq C_{a_i}(R) \times 2^{n+1}$$

and for every other agent in $C = A \setminus B$ we have:

$$C_{a_i}(R) \geq C_{a_i}(e) \times 2^{n+1}$$

Hence, e costs at least 2^{n+1} times more than R for agents in B. We call Near Exact Protocol and ask agents in B to divide the e into $2^{|C|+1}$ pieces with $\epsilon = 1/(2^{|C|+2})$, therefore each piece costs at least $C_{a_i}(e)/(2^{|C|+2}) \geq 1/2^{n+1}$ where a_i is an agent in B. By calling Oblige Protocol for these pieces and agents C, we get a partial envy-free allocation of e such that each agent in C gets at least one intact piece. Therefore, each agent a_i in B thinks that every other agent C has got a piece that costs at least $\frac{C_{a_i}(e)}{2^{n+1}} \geq C_{a_i}(R)$. Hence, even if an agent in B gets all the R, she will not envy any agent in C. However, agents in C are not happy with this assignment, since we are given a part of the chore to them, without giving anything to agents in B. Similarly R costs at least 2^{n+1} times more than e for agents in C. Again we do the same thing and run Near Exact Protocol to divide R and partially assign it to agents in B. Finally, we recursively allocate the remaining pieces of R between agents in B and remaining pieces of e between agents in C. Due to what we said, this assignment would preserve envy-freeness and allocate the whole chore.

6 Near-Exact Protocol

In this section we provide a protocol which partitions a chore R into m pieces that are almost equal for every agent. Pikhurko [17] first provided an alternative envy-free cake cutting protocol using a partitioning of a cake into m pieces that are almost equally valuable for every agent. Here we prove that a similar approach works for negative valuations and provide the full proof for completeness. In particular we prove the following lemma.

LEMMA 6.1. *Given a chore R, an integer m, and a real number $\epsilon > 0$, there exists a bounded protocol that partitions R into pieces $P_1, \ldots, P_m$, such that for every agent a, and $1 \leq i \leq m$*

$$|C_a(P_i) - C_a(R)/m| \leq \epsilon C_a(R),$$

and moreover, for agent a_n, $C_{a_n}(P_i) = C_{a_n}(R)/m$.

7 The Oblige Protocol

Oblige Protocol is called by Discrepancy Protocol to get a partial envy-free allocation such that every agent receives at least one complete piece. Discrepancy Protocol uses this property to make a set of agents dominant to the others.

Oblige Protocol gets a set of n agents and a partitioning of the chore into 2^{n+1} pieces and returns a partial envy-free allocation such that for every agent at least one piece is completely assigned to her. It asks agents a_1 to a_n respectively, asking agent a_i to set aside her 2^{i-1} most costly pieces. Then, it asks agents $a_n$to a_1 respectively, asking agent a_i to return some part of her set aside pieces to create a $2^{i-1}+1$-way tie for her smallest pieces. Finally, it asks agents a_1 to a_n respectively, to choose their smallest piece.

LEMMA 7.1. *Oblige Protocol returns a partial envy-free allocation such that each agent receives at least one complete piece.*

Proof. The protocol first asks each agent to reserve some of her largest pieces. Then, at least $2^{n+1} - (2^0 +$

$2^1 + \cdots + 2^{n-1}) \geq 2^n$ pieces remain. In lines 3 - 4 each agent is asked to create a tie between her smallest pieces using her reservation. When agent a_i is asked to create a tie between her smallest pieces, there will be at least $2^{i-1}+1$ intact pieces, thus her $(2^{i-1}+1)^{th}$ smallest piece has a value less than or equal to the value of one of the remaining intact pieces which are not greater than the values of her reserved pieces. Therefore, for each of her 1^{st}, 2^{nd},..., $2^{i-1^{th}}$ smallest pieces, she can return back some part of one of her reserved pieces to equalize them with the value of $(2^{i-1}+1)^{th}$ smallest piece. Next, each agent a_i is asked to take one of her smallest pieces. Since after agent a_i equalized her smallest pieces we have at most 2^{i-1} allocated or augmented pieces, at least one of the smallest pieces is available and she can take it.

References

[1] Haris Aziz. Computational social choice: Some current and new directions. In *IJCAI*, pages 4054–4057, 2016.

[2] Haris Aziz and Simon Mackenzie. A discrete and bounded envy-free cake cutting protocol for four agents. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 454–464. ACM, 2016.

[3] Haris Aziz and Simon Mackenzie. A discrete and bounded envy-free cake cutting protocol for any number of agents. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 416–427. IEEE, 2016.

[4] Julius B Barbanel and Alan D Taylor. Preference relations and measures in the context of fair division. *Proceedings of the American Mathematical Society*, 123(7):2061–2070, 1995.

[5] Steven Brams, Alan Taylor, and William Zwicker. A moving-knife solution to the four-person envy-free cake-division problem. *Proceedings of the American Mathematical Society*, 125(2):547–554, 1997.

[6] Steven J Brams and Alan D Taylor. An envy-free cake division protocol. *The American Mathematical Monthly*, 102(1):9–18, 1995.

[7] Steven J Brams and Alan D Taylor. *Fair Division: From cake-cutting to dispute resolution*. Cambridge University Press, 1996.

[8] Costas Busch, Mukkai S Krishnamoorthy, and Malik Magdon-Ismail. Hardness results for cake cutting. *Bulletin of the EATCS*, 86:85–106, 2005.

[9] Jeff Edmonds and Kirk Pruhs. Cake cutting really is not a piece of cake. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 271–278. Society for Industrial and Applied Mathematics, 2006.

[10] Martin Gardner. *Aha! Aha! insight*, volume 1. Scientific American, 1978.

[11] S Garfunkel. For all practical purposes social choice. *COMAP*, 1988.

[12] Egor Ianovski. Cake cutting mechanisms. *arXiv preprint arXiv:1203.0100*, 2012.

[13] David Kurokawa, John K Lai, and Ariel D Procaccia. How to cut a cake before the party ends. In *AAAI*, 2013.

[14] Claudia Lindner and Jörg Rothe. Cake-cutting: Fair division of divisible goods. In *Economics and Computation*, pages 395–491. Springer, 2016.

[15] Elisha Peterson and Francis Edward Su. Four-person envy-free chore division. *Mathematics Magazine*, 75(2):117–122, 2002.

[16] Elisha Peterson and Francis Edward Su. N-person envy-free chore division. 2009.

[17] Oleg Pikhurko. On envy-free cake division. *The American Mathematical Monthly*, 107(8):736–738, 2000.

[18] Ariel D Procaccia. Cake cutting: not just child's play. *Communications of the ACM*, 56(7):78–87, 2013.

[19] Ariel D Procaccia. Cake cutting algorithms. In *Handbook of Computational Social Choice, chapter 13*. Citeseer, 2015.

[20] Jack Robertson and William Webb. Cake-cutting algorithms: Be fair if you can. 1998.

[21] Amin Saberi and Ying Wang. Cutting a cake for five people. *AAIM*, 9:292–300, 2009.

[22] Erel Segal-Halevi, Avinatan Hassidim, and Yonatan Aumann. Waste makes haste: Bounded time protocols for envy-free cake cutting with free disposal. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 901–908. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[23] Hugo Steinhaus. The problem of fair division. *Econometrica*, 16:101–104, 1948.