Mind the Gap: Cake Cutting With Separation[1]

Edith Elkind, Erel Segal-Halevi, and Warut Suksompong

Abstract

We study the problem of fairly allocating a divisible resource, also known as cake cutting, with an additional requirement that the shares that different agents receive should be sufficiently separated from one another. This captures, for example, constraints arising from social distancing guidelines. While it is sometimes impossible to allocate a proportional share to every agent under the separation requirement, we show that the well-known criterion of maximin share fairness can always be attained. We then establish several computational properties of maximin share fairness—for instance, the maximin share of an agent cannot be computed exactly by any finite algorithm, but can be approximated with an arbitrarily small error. In addition, we consider the division of a pie (i.e., a circular cake) and show that an ordinal relaxation of maximin share fairness can be achieved.

1 Introduction

The end of the year is fast approaching, and members of a city council are busy planning the traditional New Year's fair on their city's main street. As usual, a major part of their work is to divide the space on the street among interested vendors. Each vendor naturally has a preference over potential locations, possibly depending on the proximity to certain attractions or the estimated number of customers visiting that space. Additionally, this year is different from previous years due to the social distancing guidelines issued by the government—vendors are required to be placed at least two meters apart. How should the city council allot the space so that all vendors feel fairly treated and at the same time everyone stays safe and sound under the new guidelines?

The problem of fairly allocating a heterogeneous divisible good among a set of agents in a fair manner has a long history and is commonly known as *cake cutting* [Brams and Taylor, 1996, Robertson and Webb, 1998, Procaccia, 2016]. A typical fairness criterion in cake cutting is *proportionality*, which means that each agent should receive her proportionally fair share—specifically, this amounts to $1/n$ of the agent's value for the whole cake, where n denotes the total number of agents. For any set of agents with arbitrary valuations, a proportional allocation in which each agent receives a single connected piece is guaranteed to exist. Better still, such an allocation can be found by a simple and efficient algorithm [Dubins and Spanier, 1961].

In this paper, we initiate the study of cake cutting with separation requirements. Besides the social distancing example that we mentioned, our setting captures the task of allocating machine processing time, where we need time to erase data from the previous process before the next process can be started, as well as land division, where we want space between different plots in order to avoid cross-fertilization. When separation is imposed, it is no longer the case that proportionality can always be satisfied—an extreme example is when all agents value only a common small piece of length less than the minimum gap required. A similar failure of proportionality has notably been observed in the allocation of *indivisible*

[1]An earlier version was presented at AAAI'21. The current version contains additional results as well as proofs omitted from the AAAI paper.

items (without separation), and a solution that has been proposed and widely studied in that context is *maximin share fairness* [Budish, 2011, Kurokawa et al., 2018]. Maximin share fairness requires each agent to receive her "maximin share" (MMS)—the best share that the agent can secure by dividing the items into n bundles and getting the worst bundle. In this work, we show that maximin share fairness is an appropriate substitute for proportionality in cake cutting with separation, and analyze it from an algorithmic perspective. This is one of the first uses of maximin share fairness in cake cutting (see Section 1.2).

1.1 Our Results

As is commonly done in cake cutting, we assume that the cake is represented by an interval and each agent is to be allocated a single subinterval of the cake. We further require the pieces of any two agents to be separated by distance at least s, where $s > 0$ is a given separation parameter. For the sake of exposition, we follow the convention in most of the literature and assume that the agents have additive valuations over the cake. However, as we discuss in Section 5, several of our positive results hold even for agents with arbitrary monotonic valuations.

In Section 3, we begin by proving that an allocation that gives every agent at least her maximin share always exists, meaning that maximin share fairness can be guaranteed. Such an allocation can be found by a simple algorithm provided that the algorithm knows the maximin share of each agent. Unfortunately, we show that *no* finite algorithm can compute the maximin share of an agent exactly in the standard Robertson–Webb model—this impossibility holds even when $n = 2$ and the agents have piecewise constant valuations. Nevertheless, we design an algorithm that approximates the maximin share up to an arbitrarily small error, which also allows us to compute an allocation wherein each agent obtains an arbitrarily close approximation of her maximin share. In addition, we present algorithms that decide whether the maximin share of an agent is greater than, less than, or equal to a given value r, and show that if the agents have piecewise constant valuations that are given *explicitly* as part of the input, then we can exactly compute their maximin shares, and therefore an MMS-fair allocation, in polynomial time.

In Section 4, we consider the allocation of a "pie", which is a one-dimensional circular cake and serves to model, for example, the streets around a city square, the shoreline of an island, or daily time slots for using a facility. In contrast to cake cutting, maximin share fairness cannot necessarily be guaranteed in pie cutting, and even the commonly studied cardinal multiplicative approximation cannot be obtained. Therefore, we focus instead on an *ordinal* relaxation of the maximin share, which allows each agent to partition the pie into k pieces for some parameter $k > n$. We show that when $k = n + 1$, the resulting notion—called the 1-*out-of*-$(n + 1)$ *maximin share*—can be satisfied. We then investigate computational properties of maximin share fairness in pie cutting and demonstrate several similarities and differences with cake cutting. In particular, while we can still approximate the maximin share of an agent, deciding whether the maximin share is greater than, less than, or equal to a given value r is no longer possible for any finite algorithm. A summary of our results can be found in Table 1.

1.2 Related Work

Cake cutting has long been studied by mathematicians and economists, and more recently attracted substantial interest from computer scientists, as it suggests a plethora of computational challenges. In particular, a long line of work in the artificial intelligence community in recent years has focused on cake cutting and its variants [Balkanski et al., 2014, Li et al., 2015, Brânzei et al., 2016, Alijani et al., 2017, Bei et al., 2017, Menon and Larson, 2017,

Task	Cake Cutting	Pie Cutting
Decide whether $\text{MMS}_i = r$	Yes (Cor. 3.9)	No (Thm. 4.3)
Decide whether $\text{MMS}_i > r$	Yes (Thm. 3.8)	No (Thm. 4.3)
Decide whether $\text{MMS}_i \geq r$	Yes (Thm. 3.5)	No (Thm. 4.6)
Compute the maximin share of an agent	No (Thm. 3.2)	No (Cor. 4.10)
Approximate the maximin share up to ε	Yes (Cor. 3.6)	Yes (Thm. 4.11)
Compute a maximin partition of an agent	No (Cor. 3.4)	No (Cor. 4.10)
Approximate a maximin partition up to ε	Yes (Cor. 3.6)	Yes (Thm. 4.11)

Table 1: Summary of the tasks that can and cannot be accomplished by finite algorithms in the Robertson–Webb model for cake cutting and pie cutting. All negative results hold even when the valuations of the agents are piecewise constant (but not given explicitly).

Arunachaleswaran et al., 2019b, Hosseini et al., 2020].

In order to prevent agents from receiving a collection of tiny pieces, it is often assumed that each agent must receive a connected piece [Dubins and Spanier, 1961, Stromquist, 1980, Su, 1999, Bei et al., 2012, Cechlárová and Pillárová, 2012, Cechlárová et al., 2013, Aumann and Dombb, 2015, Arunachaleswaran et al., 2019a, Goldberg et al., 2020]. Indeed, when we divide resources such as time or space, non-connected pieces (e.g., disconnected time intervals or land plots) may be hard to utilize, or even totally useless. Note that we impose the connectivity constraint not only on the allocation but also in the definition of the maximin share benchmark. Similar conventions have been used in the context of indivisible items, where the items are vertices of an undirected graph and every agent must be allocated a connected subgraph [Bouveret et al., 2017, Lonc and Truszczynski, 2020].[2]

Most previous works on cake cutting did not explicitly consider the maximin share. This is because, with additive utilities, a proportional allocation is also an MMS-fair allocation, since each agent's maximin share is always at most $1/n$ of the agent's value for the entire cake. In particular, without separation constraints, classic algorithms for proportional cake cutting [Steinhaus, 1948, Dubins and Spanier, 1961, Even and Paz, 1984] attain maximin share fairness. The maximin share becomes interesting when a proportional allocation may not exist. We know of two recent studies of the maximin share in cake cutting. Bogomolnaia and Moulin [2020] considered agents with general continuous valuations—not necessarily additive or even monotone. They showed that the maximin share is not always attainable, but the *minimax share* (i.e., the worst-case share of an agent when the items are partitioned into n bundles and the agent gets the best bundle) can always be guaranteed. Segal-Halevi [2021, Appendix B] showed that maximin share fairness can be attained when the cake is a collection of disconnected intervals and each agent should receive a connected piece.

2 Preliminaries

In cake cutting, the cake is represented by the interval $[0, 1]$. The set of agents is denoted by $N = [n]$, where $[k] := \{1, 2, \ldots, k\}$ for any positive integer k. The preference of each agent i is represented by an integrable *density function* $f_i : [0, 1] \to \mathbb{R}_{\geq 0}$, which captures how the agent values different parts of the cake. A *piece of cake* is a finite union of disjoint intervals

[2]Bei et al. [2021] explored the relations between the constrained and unconstrained versions of the maximin share in that context.

of the cake; it is said to be *connected* if it consists of a single interval. Agent i's value for a piece of cake X is given by $v_i(X) := \int_{x\in X} f_i(x)dx$. For $0 \le x \le y \le 1$, we simplify notation and write $v_i(x, y) = v_i([x, y])$. As is standard in cake cutting, we assume that the density functions are normalized so that $v_i(0, 1) = 1$ for all $i \in N$. A valuation function is said to be *piecewise constant* if it is represented by a piecewise constant density function. An *allocation* of the cake is denoted by a vector $\mathbf{A} = (A_1, \dots, A_n)$, where each A_i is a piece of cake, and A_i and A_j are disjoint for all $i \ne j$. The piece A_i is allocated to agent i. We are interested in allocations that are *connected*—each A_i is a connected piece.

Let s be a real parameter. We seek allocations in which every two pieces are separated by length at least s; we call such allocations *s-separated.* The case $s = 0$ corresponds to the classic setting (without separation), and when $s \ge \frac{1}{n-1}$ an s-separated allocation must have at least one empty piece. Therefore, from now on we assume that $s \in (0, \frac{1}{n-1})$. We define *partitions* and *s-separated partitions* in a similar manner, with the difference being that for partitions, we have a set $\mathbf{P} = \{P_1, \dots, P_n\}$ instead of a vector $\mathbf{A} = (A_1, \dots, A_n)$. The *min-value* of partition P for agent i is defined as $\min_{j=1}^{n} v_i(P_j)$. Assume without loss of generality that the pieces $P_1, \dots, P_n$ are in increasing order from left to right, and denote by $\Pi_{n,s}(x, y)$ the set that consists of all s-separated partitions of the interval $[x, y]$ (where $[x, y] \subseteq [0, 1]$). Note that an s-separated allocation or partition is incomplete since some of the cake necessarily remains unallocated. An *instance* consists of the agents, cake, density functions, and separation parameter.

A standard method for a cake-cutting algorithm to access agents' valuations is through queries in the model of Robertson and Webb [1998], which supports two types of queries:

- $\textsc{Eval}_i(x, y)$: Asks agent i to evaluate the interval $[x, y]$ and return the value $v_i(x, y)$.
- $\textsc{Cut}_i(x, \alpha)$: Asks agent i to return the leftmost point y such that $v_i(x, y) = \alpha$, or state that no such point exists.

We now define the main fairness criterion of our paper.

Definition 2.1. The *maximin share* of agent $i \in N$ with respect to an interval $[x, y] \subseteq [0, 1]$ is defined as

$$\mathrm{MMS}_i^{n,s}(x, y) := \sup_{\mathbf{P}\in\Pi_{n,s}(x,y)} \min_{j\in[n]} v_i(P_j).$$

When n and s are clear from context, we omit them from the notation and write $\mathrm{MMS}_i(x, y)$ instead of $\mathrm{MMS}_i^{n,s}(x, y)$. When $[x, y] = [0, 1]$, we further abbreviate this to MMS_i.

Let $\Pi'_{n,s}(x, y) \subseteq \Pi_{n,s}(x, y)$ be the set of s-separated partitions of $[x, y]$ such that every pair of consecutive pieces is separated by length exactly s. We claim that the definition of the maximin share can be simplified by replacing $\Pi_{n,s}(x, y)$ with $\Pi'_{n,s}(x, y)$; intuitively, for every partition in which the distance between some pair adjacent pieces is larger than s, there is a partition with at least the same min-value in which the distance between all pairs of adjacent pieces is exactly s. We also claim that the supremum in the definition can be replaced with a maximum, i.e., a maximizing partition always exists.

Proposition 2.2. *For every agent $i \in N$, it holds that*

$$\mathrm{MMS}_i^{n,s}(x, y) = \max_{\mathbf{P}\in\Pi'_{n,s}(x,y)} \min_{j\in[n]} v_i(P_j).$$

From now on, we will work with this new definition of the maximin share. An s-separated partition is said to be a *maximin partition* for agent i if every piece in the partition yields value at least $\mathrm{MMS}_i^{n,s}$. Proposition 2.2 implies that every agent has at least one maximin partition. Similarly, an s-separated allocation is said to be an *MMS-fair allocation* if every agent i receives value at least $\mathrm{MMS}_i^{n,s}$ from the allocation.

All omitted proofs can be found in Appendix A.

3 Cake Cutting

In this section, we consider cake cutting with separation, both in the Robertson–Webb query model and in a model where the agents' valuations are given explicitly.

3.1 Robertson–Webb Query Model

We begin by showing that the maximin share is an appropriate fairness criterion in our setting: it is always possible to fulfill this criterion for every agent using a quadratic number of queries in the Robertson–Webb model. Our algorithm is similar to the famous Dubins–Spanier protocol for finding proportional allocations when separation is not required [Dubins and Spanier, 1961]: we process the cake from left to right and, at each stage, allocate a subsequent piece of cake to an agent who demands the smallest piece.

Theorem 3.1. *For any instance, there exists an MMS-fair allocation. Moreover, given the maximin share of each agent, such an allocation can be computed using $O(n^2)$ queries in the Robertson–Webb model.*

Proof. We ask each agent i to mark the leftmost point x_i such that $v(0, x_i) = \mathrm{MMS}_i$. The agent who marks the leftmost x_i is allocated the piece $[0, x_i]$ (with ties broken arbitrarily); we then remove this agent along with the piece $[x_i, x_i + s]$, and recurse on the remaining agents and cake. If there is only one agent left, that agent receives all of the remaining cake. Since we make $n - j$ cut queries when there are $n - j$ agents left (and no eval queries), our algorithm uses $\sum_{j=0}^{n-1}(n - j) = O(n^2)$ queries.

We now prove the correctness of the algorithm. Consider any agent i and her maximin partition. If agent i receives the first piece allocated by the algorithm, she receives value MMS_i. Else, the allocated piece is no larger than the first piece of her maximin partition. Since the algorithm inserts a separator of length exactly s, the right endpoint of the first separator is either the same or to the left of the corresponding point in agent i's maximin partition. Applying a similar argument repeatedly, we find that if agent i is not allocated any of the first $n - 1$ pieces, then the remaining cake contains the n-th piece of the agent's partition. Hence, agent i receives a value of at least MMS_i in this case too. $\square$

The algorithm in Theorem 3.1 crucially relies on knowing the maximin share of each agent. Unfortunately, we show next that this knowledge is impossible to achieve in finite time, even if the valuations are piecewise constant but are not given explicitly as part of the input. Our result is similar in spirit to the non-finiteness results for connected envy-free cake cutting [Stromquist, 2008], equitability with connected pieces [Cechlárová and Pillárová, 2012, Brânzei and Nisan, 2017] and with arbitrary pieces [Procaccia and Wang, 2017], and average-proportionality [Segal-Halevi and Nitzan, 2019]. However, all previous impossibility results were for two or more agents with possibly different valuations. In contrast, our impossibility result is attained even for a *single* agent who wants to cut the cake into two s-separated pieces.

Theorem 3.2. *There is no algorithm that always computes $\mathrm{MMS}_i^{n,s}$ for any $s > 0$ and agent i by asking the agent a finite number of Robertson–Webb queries. This holds even when $n = 2$ and the agent's valuation is piecewise constant and strictly positive (but is not given explicitly).*

We prove the theorem by reducing from a more general problem, which may be of independent interest.

Problem FindSum1(s), where $s \in [0,1]$ is a real parameter.

Input: $g : [0,1] \to [0,1]$, a continuous monotone-increasing bijective function, specified by oracles that can answer queries of two kinds:

- Given $x \in [0,1]$, what is $g(x)$?
- Given $\alpha \in [0,1]$, what is $g^{-1}(\alpha)$?

Output: A point $x_0 \in [0, 1-s]$ for which $g(x_0) + g(x_0+s) = 1$.

Note that $g(0) = 0$ and $g(1) = 1$ due to the properties of g, and that FindSum1(s) always has a solution due to the intermediate value theorem. FindSum1(0) can be solved by one query $g^{-1}(1/2)$. Below we prove that, for any $s > 0$, FindSum1(s) cannot be solved by finitely many queries. Then, we show that FindSum1(s) can be reduced to computing $\text{MMS}_i^{2,s}$. This implies Theorem 3.2.

Lemma 3.3. *For any $s > 0$, there is no algorithm that solves* FindSum1*(s) using finitely many queries. This holds even if it is known that g is piecewise linear.*

Proof of Theorem 3.2. Let $s > 0$ be the separation parameter. We reduce FindSum1(s) to the problem of computing $\text{MMS}_i^{2,s}$: we show that, given an algorithm Alg that computes $\text{MMS}_i^{2,s}$ using at most t Robertson–Webb queries, we can solve FindSum1(s) for any function g using at most $2t+1$ queries. This is sufficient by Lemma 3.3. For each query asked by Alg, our reply will imitate a valuation v for which $v(0,x) = g(x)$, namely:

- If Alg asks $\text{Eval}_i(x,y)$, we ask $g(x)$ and $g(y)$ and reply $g(y) - g(x)$;
- If Alg asks $\text{Cut}_i(x,\alpha)$, we ask $g(x)$ and $g^{-1}(\alpha + g(x))$ and reply the latter value.

At some point, Alg stops and outputs some $r \in (0,1)$ as the value of $\text{MMS}_i^{2,s}$; at that point we ask one more query $g^{-1}(r)$ and return its result x_0 as the answer to FindSum1(s). Note that if Alg uses t queries, then we have used $2t+1$ queries.

The correctness of Alg implies that r must be the correct $\text{MMS}_i^{2,s}$ for any valuation v compatible with the query replies. This means that there exists an s-separated partition of the cake into two pieces $[0, x_0]$ and $[x_0+s, 1]$ for which $v(0,x_0) = v(x_0+s, 1) = r$. The former equality implies that $v(0,x_0) + v(0, x_0+s) = 1$. This is true, in particular, for the valuation by which we answered the queries, $v(0,x) := g(x)$. Hence, $g(x_0) = r$ so $x_0 = g^{-1}(r)$, and $g(x_0) + g(x_0+s) = 1$ so x_0 is indeed the right answer to FindSum1(s). □

Given a maximin partition of an agent, we can compute the agent's maximin share by simply taking the minimum among the agent's values for the pieces in the partition. Moreover, for two agents with identical valuations, an allocation in which each agent receives at least their (common) maximin share corresponds to a maximin partition for the common valuation. Theorem 3.2 therefore yields the following corollary, which also implies that an allocation whose existence is guaranteed by Theorem 3.1 cannot be computed without the knowledge of the agents' maximin shares.

Corollary 3.4. *There is no finite algorithm in the Robertson–Webb model that can always (a) compute a maximin partition of a single agent, or (b) compute an MMS-fair allocation for n agents. This holds even when $n = 2$ and the agents' valuations are piecewise constant (but not given explicitly).*

Despite these negative results, we show next that it is possible to get an arbitrarily good approximation of the maximin share, partition, and allocation.

Theorem 3.5. *Given an agent i and a number $r > 0$, it is possible to decide whether $\mathrm{MMS}_i \geq r$ (and, if so, compute a partition with min-value at least r for agent i) using at most n queries in the Robertson–Webb model.*

Combining the algorithm in Theorem 3.5 with binary search allows us to approximate the maximin share.

Corollary 3.6. *Given an agent i and a number $\varepsilon > 0$, it is possible to find a number r for which $\mathrm{MMS}_i - \varepsilon \leq r \leq \mathrm{MMS}_i$ (together with a partition with min-value at least r for agent i) using $O(n \log(1/\varepsilon))$ Robertson–Webb queries.*

If instead of the exact MMS_i, we are given a number $r_i \leq \mathrm{MMS}_i$ for each agent i, the algorithm for computing an MMS-fair allocation in Theorem 3.1 still computes an allocation in which agent i receives value at least r_i; the proof is essentially the same. We therefore have the following:

Corollary 3.7. *For any $\varepsilon > 0$, it is possible to compute an allocation in which every agent i receives value at least $\mathrm{MMS}_i - \varepsilon$ using at most $O(n^2 \log(1/\varepsilon))$ Robertson–Webb queries.*

Next, we consider the question of deciding whether the maximin share of an agent is *strictly greater than* a given number r. At first glance, it may seem that to this end, we can simply run the algorithm from Theorem 3.5 and answer yes exactly when the value left after $n - 1$ iterations is strictly greater than r. While this modification indeed works if the density function is positive throughout the cake, it may fail when intervals with zero value are present. Concretely, suppose that $v_i(0, 1/3) = 0.4$, $v_i(1/3, 2/3) = 0$, $v_i(2/3, 1) = 0.6$, and the value is distributed uniformly within each interval. Moreover, $s = 1/3$ and we want to decide whether $\mathrm{MMS}_i^{2,s} > 0.4$. Even though there is leftover value at the right end of the cake when we run the algorithm, which may tempt us to believe that the maximin share can go above 0.4, the zero-valued middle part in fact renders this belief false.

This example suggests a simple modification to the algorithm from Theorem 3.5: instead of marking the leftmost point such that the value of the resulting piece is r, we should mark the *rightmost* point with this property. It is easy to verify that $\mathrm{MMS}_i > r$ if and only if after executing this modified algorithm we are left with a positive-value piece. The modified algorithm requires a query $\textsc{CutRight}_i(x, \alpha)$ that returns the rightmost point y for which that $v(x, y) = \alpha$. Unfortunately, and perhaps surprisingly, $\textsc{CutRight}$ *cannot* be implemented using the queries available via the Robertson–Webb model—see Appendix B. Nevertheless, we can get around this issue by instead going over the cake from right to left. In the first iteration, we want the leftmost point x such that $v(x, 1) = r$. This is equivalent to finding the leftmost point x such that $v(0, x) = 1 - r$, which can be done with a standard $\textsc{Cut}_i(0, 1 - r)$ query.

Theorem 3.8. *Given an agent i and a number $r > 0$, it is possible to decide whether $\mathrm{MMS}_i > r$ using at most n queries in the Robertson–Webb model.*

Theorems 3.5 and 3.8 immediately imply the following:

Corollary 3.9. *Given an agent i and a number $r > 0$, it is possible to decide whether $\mathrm{MMS}_i = r$ using at most $2n$ queries in the Robertson–Webb model.*

3.2 Explicit Piecewise Constant Valuations

In this subsection, we assume that all agents have piecewise constant valuations and, moreover, these valuations are given *explicitly*. That is, for an agent $i \in N$ we are given a list of breakpoints $(p_0, p_1, \dots, p_d)$ with $p_0 = 0$, $p_d = 1$, and a list of densities $(\gamma_1, \dots, \gamma_d)$, so that for each $j \in [d]$ and each $x \in [p_{j-1}, p_j]$ the valuation function v_i of agent i satisfies $v_i(0, x) = \sum_{\ell=1}^{j-1}[\gamma_\ell \cdot (p_\ell - p_{\ell-1})] + \gamma_j \cdot (x - p_{j-1})$. Moreover, all breakpoints and densities are rational numbers, represented as fractions whose numerators and denominators are given in binary. It is straightforward to implement both types of Robertson–Webb queries in this model, so every problem that can be solved in polynomial time in the Robertson–Webb model can also be solved in polynomial time in this model. But the explicit representation offers additional benefits: we can compute the agents' maximin shares *exactly* rather than approximately.

Theorem 3.10. *Given an agent i with a piecewise constant valuation function given explicitly, we can compute $\mathrm{MMS}_i^{n,s}$ in time polynomial in the size of the input.*

At a high level, the proof of Theorem 3.10 proceeds by formulating a linear program whose solution corresponds to MMS_i. The challenge is that in order to have a linear program that returns a correct answer, we need to find out the intervals to which each endpoint of a maximin partition belongs. To accomplish this, we proceed from left to right, determining the interval for one endpoint at a time. By comparing the maximin shares between optimal subpartitions to the left and right of the potential intervals to which the next endpoint belong, we ensure that at each step of the algorithm, there exists a maximin partition whose endpoints are consistent with the intervals we have chosen.

Combined with Theorem 3.1, Theorem 3.10 implies that when agents have piecewise constant valuations given explicitly, an MMS-fair allocation can be computed efficiently (cf. Corollary 3.4).

Corollary 3.11. *For agents with piecewise constant valuations given explicitly, an MMS-fair allocation can be computed in time polynomial in the size of the input.*

4 Pie Cutting

In the canonical model of cake cutting the cake is assumed to be linear. By contrast, in this section we assume that it is circular. In other words, our resource is represented by the interval $[0, 1]$ with its two endpoints identified with each other. The respective division problem is known in the literature as *pie cutting*; its applications include dividing the shoreline of an island among its inhabitants and splitting a daily cycle for using a facility [Thomson, 2007, Brams et al., 2008, Barbanel et al., 2009].

The definitions of s-separated partitions and allocations can be readily adjusted to pie cutting—the only difference is that, due to the circular structure, there are n separators in pie cutting rather than $n - 1$ (so we assume that $s < 1/n$). Note that, since the pie is one-dimensional, distances are measured along the circumference of the pie. We denote by $\Pi_{n,s}$ the set of s-separated partitions with respect to the pie, and by $\Pi'_{n,s} \subseteq \Pi_{n,s}$ the subset of partitions for which every pair of consecutive pieces is separated by length exactly s. The maximin share can then be defined similarly to how it is defined in cake cutting (Definition 2.1); just as in Proposition 2.2, we can show that $\mathrm{MMS}_i^{n,s} = \max_{\mathbf{P} \in \Pi'_{n,s}} \min_{j \in [n]} v_i(P_j)$.

However, in pie cutting, unlike in cake cutting, an MMS-fair allocation does not necessarily exist. This is evident in the example in Figure 1, where $s > 1/4$ and $0 < \varepsilon < \min\{s - 1/4, 1/2 - s\}$, and Alice values pieces of length ε centered at the top and bottom of the pie while Bob values similar pieces on the left and right. Since $s < 1/2 - \varepsilon$, both agents have a maximin share of $1/2$. However, since the distance between any point in Alice's piece and any point in Bob's piece is at most $1/4 + \varepsilon < s$, no s-separated allocation gives both agents a positive value. Hence, no MMS-fair allocation exists.

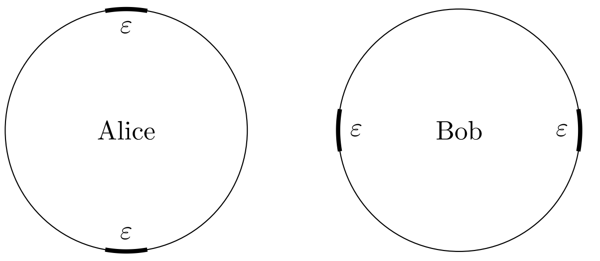


Figure 1: Example of a pie cutting instance with no MMS-fair allocation. Each of the two agents uniformly values the bold part of the pie, $s > 1/4$, and $0 < \varepsilon < \min\{s - 1/4, 1/2 - s\}$.

In fair division, a common response to the non-existence of MMS-fair allocations is to seek allocations that guarantee each agent a constant fraction of their maximin share. However, the same example shows that, in our setting, no positive fraction of the maximin share can be guaranteed. Given this negative result, it may seem unclear whether any meaningful fairness guarantee can be achieved in pie cutting with separation. Fortunately, positive results can be obtained if, instead of using *cardinal* approximations of the maximin share, we relax the criterion in an *ordinal* manner. Specifically, when each agent computes her maximin share, we allow partitioning into k pieces, where k is a parameter greater than n—we refer to the resulting notion as the 1-*out-of*-k maximin share and write $\text{MMS}_i^{k,s}$ or simply MMS_i^k for the share of agent i. Ordinal approximations were introduced by Budish [2011] in the context of indivisible item allocation. In particular, he considered the case[3] $k = n + 1$.

It turns out that this relaxation is precisely what we need for pie cutting.

Theorem 4.1. *For any pie cutting instance with n agents, there exists an allocation in which every agent i receives a piece of value at least MMS_i^{n+1}. Moreover, given the 1-out-of-$(n + 1)$ maximin share of each agent, such an allocation can be computed using $O(n^2)$ queries in the Robertson–Webb model.*

The idea behind our algorithm is similar to that of the analogous result for cake cutting (Theorem 3.1). The difference is that, because of the circular structure, when we start proceeding over the pie from a certain point (which we take to be the point 0), we may destroy one of the pieces in each agent's partition. This is why we need $n + 1$ pieces in the partition rather than n. We sometimes view the pie as an interval $[0, 1]$ and refer to 'left' and 'right' with respect to this interval.

Proof. We ask each agent i to mark the leftmost point x_i such that $v(0, x_i) = \text{MMS}_i^{n+1}$. The agent who marks the leftmost x_i is allocated the piece $[0, x_i]$ (with ties broken arbitrarily); we then remove this agent along with the piece $[x_i, x_i + s]$, and recurse on the remaining agents and pie. If there is only one agent left, we still allocate to that agent a piece worth MMS_i^{n+1} (as opposed to the entire remaining pie). Since we make $n - j$ cut queries when there are $n - j$ agents left (and no eval queries), our algorithm uses $\sum_{j=0}^{n-1}(n - j) = O(n^2)$ queries.

[3]One way to think about this relaxation is that we pretend that there are $k > n$ agents when computing the maximin share. 1-out-of-k maximin share is a special case of the ℓ-out-of-k maximin share notion introduced by Babaioff et al. [2019] and further studied by Segal-Halevi [2020], which takes the ℓ pieces of minimum value in a partition into k pieces.

We now prove the correctness of the algorithm. Consider any agent i and her 1-out-of-$(n+1)$ maximin partition. When we turn the pie into a cake by cutting at the point 0 (equivalently, the point 1), we may break one of the pieces in the partition. Nevertheless, at least n pieces remain intact. If agent i receives the first piece allocated by the algorithm, she receives value MMS_i^{n+1}. Else, the allocated piece is no larger than the first intact piece of her maximin partition. Since the algorithm inserts a separator of length exactly s, the right endpoint of the first separator is no further to the right than the left endpoint of the agent's second intact piece. Applying a similar argument repeatedly, we find that if agent i is not allocated any of the first $n-1$ pieces, then after removing the $(n-1)$-st piece and the following separator, the remaining cake contains the n-th intact piece of agent i's partition as well as a positive amount of the $(n+1)$-st piece (which may be intact or not). In particular, after allocating a piece of value MMS_i^{n+1} to the agent, the separator between the n-th and $(n+1)$-st pieces still remains. This means that there is a gap of length at least s between the first and last pieces of our allocation, and therefore the allocation is s-separated. □

Remark 4.2. Theorem 4.1 can be generalized to guarantee to each agent her ℓ-out-of-$(\ell n+1)$ maximin share, for any integer $\ell \geq 1$. As an example where this can be useful, suppose $s = 1/6$ and $n = 2$, and consider an agent who values the regions $[0, 1/30]$, $[6/30, 7/30]$, $[12/30, 13/30]$, $[18/30, 19/30]$, $[24/30, 25/30]$ uniformly at $1/5$ each (and has no value for the remaining pie). Then her 2-out-of-5 maximin share is $2/5$, which is higher than her 1-out-of-3 maximin share. On the other hand, if she values the regions $[0, 1/6]$, $[2/6, 3/6]$, $[4/6, 5/6]$ uniformly at $1/3$ each, then her 1-out-of-3 maximin share is $1/3$, which is higher than her 2-out-of-5 maximin share.

The following algorithm for the generalization is moreover *pluralistic* in that it allows each agent i to get the ℓ_i-out-of-$(\ell_i n+1)$ maximin share for her optimal ℓ_i. Reduce the pie into a cake by breaking it at an arbitrary point (say, the point 0). This destroys, for each agent i, at most a single part in her $(\ell_i n+1)$-maximin partition. Thus, at least $\ell_i n$ parts (with their adjacent separators) remain intact. A procedure similar to the one described in Theorem 4.1 then guarantees each agent at least $\ell_i n/n = \ell_i$ of these parts.

For cake cutting, there exists an algorithm that, given an agent i and a number r, decides whether $\text{MMS}_i > r$ and whether $\text{MMS}_i = r$ (Theorem 3.8 and Corollary 3.9). In contrast, for pie cutting this is not the case.

Theorem 4.3. *Fix any $k \geq 2$. For pie cutting, there is no finite algorithm in the Robertson–Webb model that can decide, for any agent i and real number r, whether $\text{MMS}_i^k > r$ or whether $\text{MMS}_i^k = r$, even when the valuation of this agent is piecewise constant (but not given explicitly).*

Theorem 4.3 leaves open the question of whether it is possible to decide whether $\text{MMS}_i^k \geq r$ for a given r. We show next that the answer to this question, too, is negative. We do so by reducing from the following problem, which may be of independent interest.

Problem HasLowValue(s, q), where $s \geq 0$ and $q \geq 0$ are real parameters.

Input: A value measure v on a pie $[0, 1]$, accessible through Cut and Eval queries.

Output: "Yes" if the pie contains an interval of length s with value at most q, i.e., there is an $x_0 \in [0, 1]$ for which $v(x_0, x_0 + s) \leq q$, where addition is done modulo 1.

Lemma 4.4. *For any real numbers $s > q > 0$, there is no algorithm that solves* HasLowValue(s,q) *using finitely many queries in the Robertson–Webb model. This holds even if the valuation v is known to be piecewise constant and strictly positive (but not given explicitly).*

Remark 4.5. The requirement $s > q > 0$ is essential for the impossibility. When $q \geq s$, the answer to HasLowValue(s,q) is always "yes". When $q = 0$, HasLowValue(s,q) can be answered using finitely many queries, using a similar idea as in the proof of Theorem 4.7 below.

Theorem 4.6. *In pie cutting, for any $k \geq 2$, there is no finite algorithm in the Robertson–Webb model that can decide, for any agent i and real number r, whether* $\text{MMS}_i^k \geq r$*, even when the valuation of this agent is piecewise constant and strictly positive (but not given explicitly).*

Proof. We prove a somewhat stronger claim: for any $r \in \left(\frac{1}{k} - s, \frac{1}{k}\right)$, there is no finite algorithm that can decide whether $\text{MMS}_i^k \geq r$. In particular, there is a whole interval of "undecidable values" rather than a single such value. For the sake of convenience, we represent the pie by the interval $[0,k]$ instead of $[0,1]$.

The proof is by reduction from HasLowValue(s,q). We show that, given an algorithm Alg that decides, for any r, whether $\text{MMS}_i^k \geq r$ on the pie $\pi_k := [0,k]$, we can decide, for any $q \in (0,s)$ and any valuation v on the pie $\pi_1 := [0,1]$, whether there exists an interval of length s and value at most q. This is sufficient by Lemma 4.4.

We run Alg with $r := \frac{1}{k} - q$; note that $r \in \left(\frac{1}{k} - s, \frac{1}{k}\right)$. For each query asked by Alg, we reply like an agent whose density function on π_k is made of k copies of the density function of v on π_1.

If there exists an interval in π_1 of length s and value at most q, then by using the k corresponding pieces of π_k as separators, the resulting partition has value at least $(1 - kq)/k = r$, so Alg answers "yes".

Conversely, suppose that no such interval exists in π_1, and consider any s-separated partition of π_k. Each separator takes up value strictly greater than q, so the remaining value outside the k separators is less than $1 - kq = kr$. Hence, at least one of the pieces in the partition has value less than r, implying that $\text{MMS}_i^k < r$; so Alg answers "no".

In both cases, the answer of Alg is the right answer to HasLowValue(s,q). □

Complementing the negative result for $r \in \left(\frac{1}{k} - s, \frac{1}{k}\right)$, we now present a positive result for $r = 1/k$. Note that we always have $\text{MMS}_i^k \leq 1/k$, and, moreover, $\text{MMS}_i^k = 1/k$ only if there is a partition where each separator has value 0. These observations turn out to be very useful for the analysis of this case.

Theorem 4.7. *For pie cutting, there exists an algorithm that, given an agent i and any $k \geq 2$, decides whether $\text{MMS}_i^k \geq 1/k$ (and if so, computes a maximin partition) using $O(k/s)$ queries in the Robertson–Webb model.*

The number of queries made by Algorithm 1 scales linearly with $1/s$. This is in contrast to Theorem 3.5 for cake cutting, where the number of queries is independent of s. We will now show that for pie cutting the number of queries must depend on s; this result holds even for $k = 2$ and $r = 1/k$.

Theorem 4.8. *Let c be any constant not depending on s. For pie cutting, there is no algorithm using at most c Robertson–Webb queries that, given an agent i, can always decide whether $\text{MMS}_i^2 \geq 1/2$, even when the agent's valuation is piecewise constant (but not given explicitly).*

At the other extreme, we show next that deciding whether the maximin share is nonzero can also be done by a finite algorithm. Moreover, unlike for $\text{MMS}_i^k \geq 1/k$, the number of queries needed to decide whether $\text{MMS}_i^k > 0$ does not depend on s. For this result we will need the assumption $s \leq \frac{1}{2k}$—this assumption means that the length of a separator is smaller than the average length of the pieces in a partition. While this is a reasonable assumption as separators are small in most applications, it remains open whether the assumption can be removed.

Theorem 4.9. *For pie cutting, there exists an algorithm that, given an agent i and any $k \geq 2$ and $s \leq \frac{1}{2k}$, decides whether $\text{MMS}_i^k > 0$ using $O(k)$ queries in the Robertson–Webb model.*

We now turn to the problem of computing the maximin share and a maximin partition of a pie. Theorem 4.3 obviously rules out the possibility of exact computation:

Corollary 4.10. *Fix any $k \geq 2$. For pie cutting, there is no finite algorithm in the Robertson–Webb model that, given an agent i, can either (a) compute MMS_i^k, or (b) compute a maximin partition into k pieces for i. This holds even when the valuation of this agent is piecewise constant (but not given explicitly).*

Despite these negative results, we show that it is possible to approximate the maximin share of an agent up to an arbitrary error. The idea is to mark points on the pie so that any piece between two adjacent marks has value at most $\varepsilon/2$, and, for each piece between two (not necessarily adjacent) marks, try to construct an s-separated partition with min-value equal to the value of this piece, by means of a greedy algorithm.

Theorem 4.11. *Fix any $k \geq 2$. For pie cutting, given an agent i and a number $\varepsilon > 0$, it is possible to find a number r such that $\text{MMS}_i^k - \varepsilon \leq r \leq \text{MMS}_i^k$, along with an s-separated partition with min-value r, using $O(1/\varepsilon)$ queries in the Robertson–Webb model.*

5 Conclusion and Future Work

In this paper, we have initiated the study of cake cutting under separation requirements, which capture scenarios including data erasure in machine processing, cross-fertilization prevention in land allocation, as well as social distancing. We established several existence and computational results on maximin share fairness—overall, our results indicate that maximin share fairness is an appropriate substitute for proportionality in this setting.

Interestingly, several of our positive results, including Theorems 3.1, 3.5, 3.8, and 4.1, do not rely on the assumption that valuations are additive (which is standard in the cake-cutting literature [Procaccia, 2016]) and work for agents with arbitrary monotonic valuations. This observation reveals another significant advantage of maximin share fairness over proportionality: when valuations are not necessarily additive, even in the absence of separation requirements, no multiplicative approximation of proportionality can be guaranteed.[4]

At a higher level, separation requirements represent one type of constraints that arise in a number of applications of cake cutting. In other applications, it may be desirable to limit the amount of cake that certain agents receive, or ensure that the cake is allocated to agents in a given order. Examining the interplay between such constraints and fairness considerations is an important direction that will likely lead to fruitful research.

[4]To see this, consider agents with identical valuations such that the value of a piece is defined by a non-decreasing function $f(\ell)$ that depends only on the length ℓ of the piece. Even without separation, in any allocation, at least one of the agents will receive value at most $f(1/n)$, which can be 0 (or, if f is required to be strictly increasing, arbitrarily close to 0).

Acknowledgments

This work was partially supported by the European Research Council (ERC) under grant number 639945 (ACCORD), by the Israel Science Foundation under grant number 712/20, and by an NUS Start-up Grant. Part of the work was done while the third author was a postdoctoral researcher at the University of Oxford. We would like to thank the anonymous reviewers of AAAI 2021 and COMSOC 2021 for their valuable comments.

References

Reza Alijani, Majid Farhadi, Mohammad Ghodsi, Masoud Seddighin, and Ahman S. Tajik. Envy-free mechanisms with minimum number of cuts. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, pages 312–318, 2017.

Eshwar Ram Arunachaleswaran, Siddharth Barman, Rachitesh Kumar, and Nidhi Rathi. Fair and efficient cake division with connected pieces. In *Proceedings of the 15th Conference on Web and Internet Economics (WINE)*, pages 57–70, 2019a.

Eshwar Ram Arunachaleswaran, Siddharth Barman, and Nidhi Rathi. Fair division with a secretive agent. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*, pages 1732–1739, 2019b.

Yonatan Aumann and Yair Dombb. The efficiency of fair division with connected pieces. *ACM Transactions on Economics and Computation*, 3(4):23:1–23:16, 2015.

Moshe Babaioff, Noam Nisan, and Inbal Talgam-Cohen. Fair allocation through competitive equilibrium from generic incomes. In *Proceedings of the 2nd ACM Conference on Fairness, Accountability, and Transparency (ACM FAT*)*, page 180, 2019.

Eric Balkanski, Simina Brânzei, David Kurokawa, and Ariel D. Procaccia. Simultaneous cake cutting. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence (AAAI)*, pages 566–572, 2014.

Julius B. Barbanel, Steven J. Brams, and Walter Stromquist. Cutting a pie is not a piece of cake. *American Mathematical Monthly*, 116(6):496–514, 2009.

Xiaohui Bei, Ning Chen, Xia Hua, Biaoshuai Tao, and Endong Yang. Optimal proportional cake cutting with connected pieces. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI)*, pages 1263–1269, 2012.

Xiaohui Bei, Ning Chen, Guangda Huzhang, Biaoshuai Tao, and Jiajun Wu. Cake cutting: Envy and truth. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3625–3631, 2017.

Xiaohui Bei, Ayumi Igarashi, Xinhang Lu, and Warut Suksompong. The price of connectivity in fair division. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI)*, 2021. Forthcoming.

Anna Bogomolnaia and Hervé Moulin. Guarantees in fair division: General or monotone preferences. *arXiv preprint 1911.10009*, 2020.

Sylvain Bouveret, Katarína Cechlárová, Edith Elkind, Ayumi Igarashi, and Dominik Peters. Fair division of a graph. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 135–141, 2017.

Steven J. Brams and Alan D. Taylor. *Fair Division: From Cake-Cutting to Dispute Resolution.* Cambridge University Press, 1996.

Steven J. Brams, Michael A. Jones, and Christian Klamler. Proportional pie-cutting. *International Journal of Game Theory*, 36(3–4):353–367, 2008.

Simina Brânzei and Noam Nisan. The query complexity of cake cutting. *arXiv preprint arXiv:1705.02946*, 2017.

Simina Brânzei, Ioannis Caragiannis, David Kurokawa, and Ariel D. Procaccia. An algorithmic framework for strategic fair division. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI)*, pages 418–424, 2016.

Eric Budish. The combinatorial assignment problem: Approximate competitive equilibrium from equal incomes. *Journal of Political Economy*, 119(6):1061–1103, 2011.

Katarína Cechlárová and Eva Pillárová. On the computability of equitable divisions. *Discrete Optimization*, 9(4):249–257, 2012.

Katarína Cechlárová, Jozef Doboš, and Eva Pillárová. On the existence of equitable cake divisions. *Information Sciences*, 228:239–245, 2013.

Lester E. Dubins and Edwin H. Spanier. How to cut a cake fairly. *American Mathematical Monthly*, 68(1):1–17, 1961.

Shimon Even and Azaria Paz. A note on cake cutting. *Discrete Applied Mathematics*, 7(3): 285–296, 1984.

Paul W. Goldberg, Alexandros Hollender, and Warut Suksompong. Contiguous cake cutting: Hardness results and approximation algorithms. *Journal of Artificial Intelligence Research*, 69:109–141, 2020.

Hadi Hosseini, Ayumi Igarashi, and Andrew Searns. Fair division of time: Multi-layered cake cutting. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 182–188, 2020.

David Kurokawa, Ariel D. Procaccia, and Junxing Wang. Fair enough: Guaranteeing approximate maximin shares. *Journal of the ACM*, 64(2):8:1–8:27, 2018.

Minming Li, Jialin Zhang, and Qiang Zhang. Truthful cake cutting mechanisms with externalities: Do not make them care for others too much! In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 589–595, 2015.

Zbigniew Lonc and Miroslaw Truszczynski. Maximin share allocations on cycles. *Journal of Artificial Intelligence Research*, 69:613–655, 2020.

Vijay Menon and Kate Larson. Deterministic, strategyproof, and fair cake cutting. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 352–358, 2017.

Ariel D. Procaccia. Cake cutting algorithms. In Felix Brandt, Vincent Conitzer, Ulle Endriss, Jérôme Lang, and Ariel D. Procaccia, editors, *Handbook of Computational Social Choice*, chapter 13, pages 311–329. Cambridge University Press, 2016.

Ariel D. Procaccia and Junxing Wang. A lower bound for equitable cake cutting. In *Proceedings of the 18th ACM Conference on Economics and Computation (EC)*, pages 479–495, 2017.

Jack Robertson and William Webb. *Cake-Cutting Algorithms: Be Fair if You Can.* Peters/CRC Press, 1998.

Erel Segal-Halevi. Competitive equilibrium for almost all incomes: Existence and fairness. *Autonomous Agents and Multi-Agent Systems*, 34(1):26:1–26:50, 2020.

Erel Segal-Halevi. Fair multi-cake cutting. *Discrete Applied Mathematics*, 291:15–35, 2021.

Erel Segal-Halevi and Shmuel Nitzan. Fair cake-cutting among families. *Social Choice and Welfare*, 53(4):709–740, 2019.

Hugo Steinhaus. The problem of fair division. *Econometrica*, 16(1):101–104, 1948.

Walter Stromquist. How to cut a cake fairly. *American Mathematical Monthly*, 87(8): 640–644, 1980.

Walter Stromquist. Envy-free cake divisions cannot be found by finite protocols. *Electronic Journal of Combinatorics*, 15:#R11, 2008.

Francis Edward Su. Rental harmony: Sperner's lemma in fair division. *American Mathematical Monthly*, 106(10):930–942, 1999.

William Thomson. Children crying at birthday parties. Why? *Economic Theory*, 31(3): 501–521, 2007.

Edith Elkind
University of Oxford
Oxford, UK
Email: `elkind@cs.ox.ac.uk`

Erel Segal-Halevi
Ariel University
Ariel, Israel
Email: `erelsgl@gmail.com`

Warut Suksompong
National University of Singapore
Singapore
Email: `warut@comp.nus.edu.sg`

A Omitted Proofs

A.1 Proof of Proposition 2.2

Fix a partition $\mathbf{P} \in \Pi_{n,s}(x,y)$. If some pair of consecutive pieces of $\mathbf{P}$ is separated by length more than s, then by extending some pieces so that every pair of consecutive pieces is separated by length exactly s, we obtain another partition $\mathbf{P}' \in \Pi'_{n,s}(x,y)$ such that $P_j \subseteq P'_j$ for all $j \in [n]$. It follows that $\min_{j\in[n]} v_i(P'_j) \geq \min_{j\in[n]} v_i(P_j)$, so in Definition 2.1 we may replace $\Pi_{n,s}(x,y)$ with $\Pi'_{n,s}(x,y)$.

Next, we show that we can also replace sup with max. Define

$$B := \{(x_1,\dots,x_{n-1}) \mid x \leq x_1 \leq \dots \leq x_{n-1} \leq y-s, \text{ and } x_i + s \leq x_{i+1} \ \forall i \in [n-2]\}.$$

Let $g_i : B \to [0,1]$ be a function defined by

$$g_i(x_1,\dots,x_{n-1}) := \min\{v_i(x,x_1), v_i(x_1+s,x_2),\dots,v_i(x_{n-1}+s,y)\}.$$

Since B is a closed and bounded subset of $\mathbb{R}^{n-1}$, the Heine–Borel theorem implies that it is compact. Moreover, since v_i is the integral of an integrable function, each of the n functions inside the minimum operator is continuous. This means that g_i, which is a minimum of continuous functions, is continuous too. Hence, the extreme value theorem for functions of several variables implies that g_i attains a maximum in B. It follows that

$$\begin{aligned}
\mathrm{MMS}_i^{n,s}(x,y) &= \sup_{\mathbf{P}\in\Pi'_{n,s}(x,y)} \min_{j\in[n]} v_i(P_j) \\
&= \sup_{(x_1,\dots,x_{n-1})\in B} g_i(x_1,\dots,x_{n-1}) \\
&= \max_{(x_1,\dots,x_{n-1})\in B} g_i(x_1,\dots,x_{n-1}) \\
&= \max_{\mathbf{P}\in\Pi'_{n,s}(x,y)} \min_{j\in[n]} v_i(P_j),
\end{aligned}$$

as claimed.

A.2 Proof of Lemma 3.3

Assume for contradiction that a finite algorithm exists. We will show how an adversary can answer the queries made by the algorithm in such a way that after any finite number of queries, for any value of x_0 that the algorithm may output, there exists a piecewise-linear function g consistent with the adversary's answers for which x_0 is not the right answer for the algorithm. This is sufficient to obtain the desired contradiction.

During the run, there is always a finite set of points $x \in [0,1]$ for which the algorithm knows the value of $g(x)$; we say that such points are *recorded*. Initially, only points 0 and 1 are recorded: since g is a bijection and it is monotone-increasing, $g(0) = 0$ and $g(1) = 1$. Given a point $x \in [0,1]$, we denote by x_- the largest recorded point that is at most x, and by x_+ the smallest recorded point that is at least x. If x itself is recorded, then $x_- = x_+ = x$.

When asked "given x, what is $g(x)$?", if x is recorded then the adversary replies $g(x)$; else, the adversary chooses a value for $g(x)$ satisfying the following properties:

(i) Monotonicity is preserved, i.e., $g(x_-) < g(x) < g(x_+)$.

(ii) If the point $x + s$ is recorded, then $g(x) \neq 1 - g(x+s)$.

(iii) If the point $x - s$ is recorded, then $g(x) \neq 1 - g(x-s)$.

Since condition (i) allows infinitely many values to choose from, and each of the conditions (ii) and (iii) rules out at most one value, the adversary can make a choice satisfying these conditions.

When asked "given α, what is $g^{-1}(\alpha)$?", if there is a recorded point x such that $g(x) = \alpha$, then the adversary replies x; else, the adversary chooses a point x satisfying the following properties:

(i) Monotonicity is preserved, i.e., $g(x_-) < \alpha < g(x_+)$.

(ii) None of the points $x - s$, $x + s$, and x is recorded.

Again, the former condition allows infinitely many points, and the latter forbids only a finite number of them.

Since the algorithm is finite, it must eventually return some number $x_0 \in [0, 1 - s]$. Now, in order to falsify the algorithm's answer, the adversary voluntarily answers two more queries by the same rules as above: $g(x_0)$ and $g(x_0 + s)$. Now both x_0 and $x_0 + s$ are recorded. The answering rules guarantee that $g(x_0) + g(x_0 + s) \neq 1$, so x_0 cannot be a correct solution to FindSum1(s).

Finally, to complete the function g, the adversary simply connects every pair of consecutive recorded points linearly.

A.3 Proof of Theorem 3.5

The idea is similar to that in Theorem 3.1. Ask the agent to mark the leftmost point x such that $v(0, x) = r$, make $[0, x]$ one of the pieces in a potential partition, add a separator $[x, x + s]$, and repeat starting from $x + s$. If there is still value at least r left after $n - 1$ iterations, answer yes; else, answer no. It is clear that at most n queries are used.

If the algorithm answers yes, then it finds a partition with value at least r, so $\text{MMS}_i \geq r$. Conversely, suppose that $\text{MMS}_i \geq r$, and consider a maximin partition. The right endpoint of the first piece in this partition is either the same or to the right of our first marked point x. In addition, since our algorithm inserts a separator of length exactly s, the right endpoint of our first separator is no further to the right than the corresponding point in the maximin partition. Applying a similar argument $n - 1$ times, we find that the right endpoint of our $(n - 1)$-st separator is no further to the right than the corresponding point in the maximin partition. Since the final piece of the partition has value at least MMS_i, our remaining piece also has value at least $\text{MMS}_i \geq r$. Hence the algorithm answers yes, as claimed.

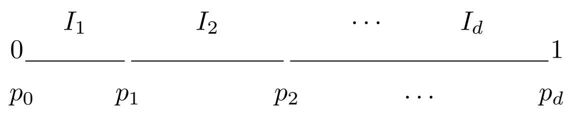
A.4 Proof of Theorem 3.8

Ask the agent to mark the leftmost point x such that $v(0, x) = 1 - r$, make $[x, 1]$ one of the pieces in a potential partition, add a separator $[x - s, x]$, and repeat going from $x - s$ leftwards. If there is still cake left after n pieces have been created, answer yes; else, answer no.

If we cannot create n pieces or if there is no cake left once we have created n pieces, then by an argument similar to those in Theorems 3.1 and 3.5, the maximin share cannot be greater than r. Suppose now that there is cake left after n pieces have been created. We will show that the partition can be modified so that it has min-value more than r. First, we extend the left end of the leftmost piece by an arbitrarily small positive amount so that the piece has value greater than r—this is possible by definition of the left endpoint—and shrink the right end so that the value of the piece is still more than r. We then repeat this process for subsequent pieces, with the modification that instead of extending the left end of each piece by an arbitrarily small amount, we extend it by no more than the amount by which we shrank the right end of the preceding piece; this ensures that the separation between any two consecutive pieces is still at least s. The resulting partition is therefore s-separated and has min-value greater than r, so $\mathrm{MMS}_i > r$, as claimed.

A.5 Proof of Theorem 3.10

Let v be the piecewise constant valuation function of agent i, described by a list of breakpoints $(p_0, p_1, \ldots, p_d)$ with $p_0 = 0$, $p_d = 1$, and a list of densities $(\gamma_1, \ldots, \gamma_d)$. For readability, we set $I_j := [p_{j-1}, p_j]$ for each $j \in [d]$, as illustrated below:



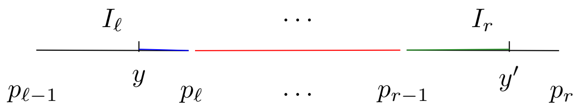
We also write

$$c^* := \mathrm{MMS}_i^{n,s}, \quad w_j := v(0, p_j) = \sum_{\ell=1}^{j} \gamma_\ell \cdot (p_\ell - p_{\ell-1})$$

Since we can check whether $c^* > 0$ using Theorem 3.8, we assume from now on that this is the case. Given two points $y \le y'$ with $y \in I_\ell$ and $y' \in I_r$, the value $v(y, y')$ is a linear function of y and y':

$$v(y, y') = (p_\ell - y)\gamma_\ell + (w_{r-1} - w_\ell) + (y' - p_{r-1})\gamma_r \tag{1}$$

This is illustrated below:



In what follows, we depart from the notation used in the remainder of the paper and represent an s-separated partition of $[0,1]$ into n parts as $(x_0, x_1, \ldots, x_n)$ where $x_0 = -s$ and $x_n = 1$, so that the k-th part of the partition is given by $[x_{k-1} + s, x_k]$.

For each $t \in [0,1]$, each $k \in [n]$, and each list of intervals $\mathcal{I}_k = (I_{\ell(1)}, I_{r(1)}, \ldots, I_{\ell(k)}, I_{r(k)})$ such that $\ell(1) \le r(1) \le \cdots \le \ell(k) \le r(k)$ and $0 \in I_{\ell(1)}$, $t \in I_{r(k)}$, consider the following linear program, with variables $x_0, x_1, \ldots, x_k, c$:

Program LP$_k(\mathcal{I}_k, t)$:

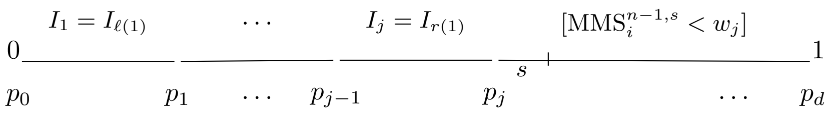
$$\begin{aligned}
\max \quad & c \\
& \text{subject to} \\
& x_0 = -s, \qquad x_k = t \\
& x_{q-1} + s \in I_{\ell(q)}, \quad x_q \in I_{r(q)} \qquad \text{for all } q \in [k] && (2) \\
& x_{q-1} + s \le x_q \qquad \text{for all } q \in [k] \\
& v(x_{q-1} + s, x_q) \ge c \qquad \text{for all } q \in [k] && (3)
\end{aligned}$$

Note that $\mathrm{LP}_k(\mathcal{I}_k, t)$ is indeed a linear program; in particular, constraints (2) are linear, because the endpoints of each I_j are known, and constraints (3) are linear because, by (1), the function $v(y, y')$ is a linear function of y and y' as long as the intervals containing y and y' are known. Every feasible solution of this linear program corresponds to an s-separated partition of $[0, t]$ into k parts, where the beginning and the end of the q-th part (i.e., $x_{q-1} + s$ and x_q in the linear program), $1 \le q \le k$, are contained in the intervals $I_{\ell(q)}$ and $I_{r(q)}$, respectively; thus, the solution of this linear program is the maximum value among all such partitions. It follows that we can compute MMS_i by solving a linear program as long as we know the 'correct' intervals for both endpoints of each part of an s-separated maximin partition. We now show how to identify such intervals in polynomial time.

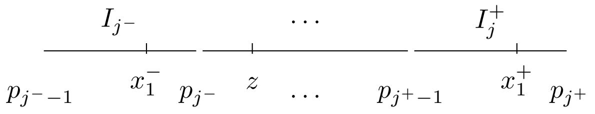
Given $k \in [n]$, a list of intervals $\mathcal{I}_k = (I_{\ell(q)}, I_{r(q)})_{q\in[k]}$ and a partition $(x_0, \dots, x_n)$ of $[0, 1]$, we will say that $(x_0, \dots, x_n)$ and $\mathcal{I}_k$ are *consistent* if $x_{q-1} + s \in I_{\ell(q)}$ and $x_q \in I_{r(q)}$ for each $q \in [k]$. If $(x_0, \dots, x_n)$ is an s-separated maximin partition of $[0, 1]$, and $\mathcal{I}_n = (I_{\ell(q)}, I_{r(q)})_{q\in[n]}$ is consistent with $(x_0, \dots, x_n)$, then the solution of the linear program $\mathrm{LP}_n(\mathcal{I}_n, 1)$ is c^*.

To build a list $\mathcal{I}_n$ that is consistent with some s-separated maximin partition, we proceed inductively. First, we construct a list $\mathcal{I}_1 = (I_{\ell(1)}, I_{r(1)})$ that is consistent with some s-separated maximin partition. Then, for each $1 < k \le n$ we extend the list $\mathcal{I}_{k-1} = (I_{\ell(q)}, I_{r(q)})_{q\in[k-1]}$ to a list $\mathcal{I}_k = (I_{\ell(q)}, I_{r(q)})_{q\in[k]}$ so that $\mathcal{I}_k$ is also consistent with some s-separated maximin partition.

Base case We compute $\mathcal{I}_1 = (I_{\ell(1)}, I_{r(1)})$ as follows. Since the first part of any partition starts at 0, we set $\ell(1) = 1$. To compute $r(1)$, for each $j \in \{0, \dots, d\}$, set $w_j := v(0, p_j)$, and use the algorithm from Theorem 3.5 to check[5] whether $w_j \le \mathrm{MMS}_i^{n-1,s}(p_j + s, 1)$. Pick the first j for which the answer is "no", and set $I_{r(1)} = I_j$, as illustrated below:



To see that this approach is correct, consider two s-separated maximin partitions of $[0, 1]$, which we denote by $(x_0^-, \dots, x_n^-)$ and $(x_0^+, \dots, x_n^+)$: these partitions are chosen so that for every s-separated maximin partition $(x_0, \dots, x_n)$ of $[0, 1]$ we have $x_1^- \le x_1 \le x_1^+$. (These two partitions may coincide.) Since $c^* > 0$, we have $0 < x_1^-$ and $x_1^+ < 1$. Suppose that $p_{j^- -1} < x_1^- \le p_{j^-}$ and $p_{j^+ -1} \le x_1^+ < p_{j^+}$ for some $j^-, j^+ \in [d]$. For every $z \in [x_1^-, x_1^+]$, the partition $(x_0^-, z, x_2^+, \dots, x_n^+)$ is also an s-separated maximin partition of $[0, 1]$. Thus, for every j such that $j^- \le j \le j^+$, there exists an s-separated maximin partition of $[0, 1]$ such that the right endpoint of the first part lies in I_j, i.e., any such choice of j is suitable for $r(1)$; we will argue that our algorithm selects $r(1)$ so that $j^- \le r(1) \le j^+$.



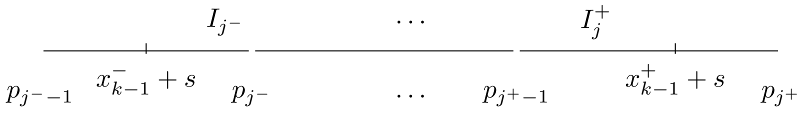
Indeed, by our choice of x_1^- we have $v(0, x_1^-) = c^*$: if $v(0, x_1^-) < c^*$, then the value of the first part is less than c^*, and if $v(0, x_1^-) > c^*$, we can find an $x < x_1^-$ with $v(0, x) = c^*$, a contradiction with our choice of x_1^-. By the same argument, $v(0, x) < v(0, x_1^-) = c^*$ for every $x < x_1^-$ and hence $v(0, p_{j'}) < c^*$ for every $j' < j^-$. On the other hand, for $j' < j^-$, $[p_{j'} + s, 1]$

[5]Even though the algorithm in Theorem 3.5 is designed for $\mathrm{MMS}_i^{n,s}(0, 1)$, it can be easily adapted to our task by allowing $n - 1$ pieces in the partition and considering the interval $[p_j + s, 1]$ instead of $[0, 1]$. Similar statements hold for other applications of the algorithm in the remainder of this proof.

is a superset of $[x_1^- + s, 1]$ and hence $\mathrm{MMS}_i^{n-1,s}(p_{j'} + s, 1) \geq c^*$. Thus, $r(1) \geq j^-$. By a similar argument, $v(0, p_{j^+}) \geq v(0, x_1^+) \geq c^*$. Further, if $\mathrm{MMS}_i^{n-1,s}(p_{j^+} + s, 1) \geq c^*$, then, by combining the corresponding maximin partition with $[0, p_{j^+}]$, we obtain an s-separated partition of $[0, 1]$ into n parts such that the value of each part is at least c^* and the first part ends at $p_{j^+} > x_1^+$, a contradiction with our choice of x_1^+. Thus, no such partition of $[p_{j^+}, 1]$ exists and hence our algorithm selects $r(1) \leq j^+$. This concludes the proof for $k = 1$.

Inductive step Now, fix an integer k with $1 < k \leq n$, and suppose we have already constructed a list $\mathcal{I}_{k-1} = (I_{\ell(q)}, I_{r(q)})_{q \in [k-1]}$ that is consistent with some s-separated maximin partition of $[0, 1]$. Our goal is to compute $I_{\ell(k)}, I_{r(k)}$ so that the list $\mathcal{I}_k = (I_{\ell(q)}, I_{r(q)})_{q \in [k]}$ is consistent with some s-separated maximin partition of $[0, 1]$.

We start with $I_{\ell(k)}$. Among all s-separated maximin partitions that are consistent with $\mathcal{I}_{k-1}$, consider a partition with the smallest value of x_{k-1} and one with the largest value of x_{k-1}; we denote these partitions by $(x_0^-, \dots, x_n^-)$ and $(x_0^+, \dots, x_n^+)$, respectively. Since $c^* > 0$, we have $x_{k-1}^+ + s < 1$. Suppose that $p_{j^- - 1} < x_{k-1}^- + s \leq p_{j^-}$ and $p_{j^+ - 1} \leq x_{k-1}^+ + s < p_{j^+}$ for some $j^-, j^+ \in [d]$. Note that for every z such that $x_{k-1}^- \leq z \leq x_{k-1}^+$ it holds that $(x_0^- \dots, x_{k-2}^-, z, x_k^+, \dots, x_n^+)$ is an s-separated maximin partition. Therefore, for every j such that $j^- \leq j \leq j^+$, there exists an s-separated maximin partition $(y_0, \dots, y_n)$ of $[0, 1]$ that is consistent with $\mathcal{I}_{k-1}$ and satisfies $y_{k-1} + s \in I_j$. Hence, any j in this range is a suitable choice for $\ell(k)$.



Let $\mathcal{L}$ be the collection of all intervals I_j that have a non-empty intersection with $[p_{r(k-1)-1} + s, p_{r(k-1)} + s]$, where $r(k-1)$ is the index of the rightmost interval in $\mathcal{I}_{k-1}$. For every interval $I_j \in \mathcal{L}$, we solve $\mathrm{LP}_{k-1}(\mathcal{I}_{k-1}, p_j - s)$; let c' be the solution of this linear program. We then check whether $\mathrm{MMS}_i^{n-k+1,s}(p_j, 1) \geq c'$ using the algorithm from Theorem 3.5, and set $\ell(k)$ to be the smallest j for which this is not the case.

To see the correctness of our approach, first observe that it suffices to restrict our attention to intervals in $\mathcal{L}$. Indeed, for every maximin partition $(x_0, \dots, x_n)$ that is consistent with $\mathcal{I}_{k-1}$ we have $x_{k-1} \in I_{r(k-1)}$, and hence $p_{r(k-1)-1} + s \leq x_{k-1} + s \leq p_{r(k-1)} + s$. Thus, $x_{k-1} + s$ has to be contained in an interval I_j such that $I_j \cap [p_{r(k-1)-1} + s, p_{r(k-1)} + s] \neq \emptyset$.

Next, we will argue that (a) $\ell(k) \geq j^-$, and (b) $\ell(k) \leq j^+$. To prove (a), consider some $j' < j^-$. We claim that the solution of $\mathrm{LP}_{k-1}(\mathcal{I}_{k-1}, p_{j'} - s)$ is strictly less than c^*. Indeed, otherwise there is an s-separated partition of $[0, p_{j'} - s]$ into $k - 1$ parts that is consistent with $\mathcal{I}_{k-1}$ and such that the value of each part is at least c^*; since $p_{j'} \leq p_{j^- - 1} < x_{k-1}^- + s$, combining this partition with $[p_{j'}, x_k^-], [x_k^- + s, x_{k+1}^-], \dots, [x_{n-1}^- + s, 1]$, we obtain a maximin partition of $[0, 1]$ that is consistent with $\mathcal{I}_{k-1}$, a contradiction with our choice of j^-. Also, we claim that $\mathrm{MMS}_i^{n-k+1,s}(p_{j'}, 1) \geq c^*$: again, this is witnessed by $[p_{j'}, x_k^-], [x_k^- + s, x_{k+1}^-], \dots, [x_{n-1}^- + s, 1]$. Hence, our algorithm will not set $\ell(k) = j'$.

To prove (b), observe that since $p_{j^+} - s > x_{k-1}^+$, we have that $(x_0^+, x_1^+, \dots, x_{k-2}^+, p_{j^+} - s, c^*)$ is a feasible solution for $\mathrm{LP}_{k-1}(\mathcal{I}_{k-1}, p_{j^+} - s)$, i.e., the solution of this LP is at least c^*. Now, if $\mathrm{MMS}_i^{n-k+1,s}(p_{j^+}, 1) \geq c^*$, we can combine the corresponding partition with $[0, x_1^+], [x_1^+ + s, x_2^+], \dots, [x_{k-2}^+ + s, p_{j^+} - s]$ to obtain an s-separated maximin partition of $[0, 1]$ where the k-th part starts at p_{j^+}, a contradiction with our choice of j^+. Thus, we have $\ell(k) \leq j^+$.

Combining (a) and (b), we conclude that $j^- \leq \ell(k) \leq j^+$, so our choice of $\ell(k)$ works.

To compute $I_{r(k)}$, we proceed in a similar fashion. Specifically, let $\mathcal{R}$ be the collection of intervals $(I_j)_{\ell(k) \leq j \leq d}$. For each interval I_j in this collection, let $\mathcal{I}_k(j)$ be the list obtained by taking $\mathcal{I}_{k-1}$ and appending $I_{\ell(k)}$ (which we have computed in the previous step) and I_j to it. We then solve $\mathrm{LP}_k(\mathcal{I}_k(j), p_j)$; let c' be the solution of this linear program. We check whether $\mathrm{MMS}_i^{n-k,s}(p_j + s, 1) \geq c'$, pick the smallest j for which this is not the case, and set $I_{r(k)} = I_j$.

The proof of correctness for this case is similar. We know that the set of maximin partitions $(x_0, \ldots, x_n)$ that are consistent with $\mathcal{I}_{k-1}$ and satisfy $x_{k-1} + s \in I_{\ell(k)}$ is not empty. Among all such partitions, consider one with the minimum x_k and one with the maximum x_k. Let I_{j^-} and I_{j^+} be the respective intervals containing the k-th cut x_k; we claim that $j^- \leq r(k) \leq j^+$. Both inequalities are proved in exactly the same way as for $\ell(k)$. This completes the inductive argument.

Both for $I_{\ell(k)}$ and for $I_{r(k)}$, we can speed up the search for the appropriate j by using binary search on $\mathcal{L}$ (respectively, $\mathcal{R}$) rather than checking each interval in that list; with this modification, we can identify each interval in $\mathcal{I}_n$ by solving $O(\log n)$ linear programs.

To see that our algorithm runs in polynomial time, it remains to observe that, to find the maximin share of agent i, we first need to identify all intervals in $\mathcal{I}_n$, and then solve the resulting linear program. Thus, we need to solve $O(n \log n)$ linear programs, and the size of each linear program is polynomial in the size of the input. The remaining steps of the algorithm (such as invoking the algorithm from Theorem 3.5) can also be implemented efficiently.

A.6 Proof of Theorem 4.3

Assume for contradiction that such an algorithm exists, and take $r = \frac{1}{k} - s$. We show how an adversary can answer the queries of the algorithm in such a way that after a finite number of queries, there exists a piecewise constant valuation function consistent with the answers for which $\mathrm{MMS}_i^k > r$, but also one for which $\mathrm{MMS}_i^k = r$.

The adversary records any point that appears in a query or its own answer, and answers queries as if the valuation is uniform throughout the pie—that is, for any two consecutive recorded points on the pie, if the interval between them has length t, then it also has value t. Suppose that some finite number of queries have been answered in this manner, and consider the following two possibilities:

Possibility 1: The entire valuation function is uniform. For any s-separated partition, the k pieces have total length at most $1 - ks$, so one of the pieces has length (and value) at most $\frac{1-ks}{k} = r$. On the other hand, there exists an s-separated partition such that each of the k pieces has length r. Hence $\mathrm{MMS}_i^k = r$ in this case.

Possibility 2: Consider all s-separated k-partitions in which each of the k pieces has length r. Among all such partitions, choose a partition $\mathbf{P}$ for which none of the $2k$ endpoints of the pieces coincides with any recorded point; since there are infinitely many partitions and only a finite number of them are forbidden, this choice is possible. If a piece or a separator does not contain a recorded point, record an arbitrary point in its interior. This ensures that each interval between two recorded points contains at most one endpoint of $\mathbf{P}$. Now, for each interval I of length z containing an endpoint of $\mathbf{P}$, distribute a value of z uniformly within the intersection of I with the associated piece of $\mathbf{P}$, so the intersection of I and the associated separator has value zero. For the remaining intervals, their value (which is equal to their length) is distributed uniformly within the interval. The resulting valuation function is piecewise constant, and each of the k pieces of $\mathbf{P}$ has value strictly greater than r. Hence $\mathrm{MMS}_i^k > r$.

We conclude that a finite algorithm cannot distinguish between the case $\mathrm{MMS}_i^k > r$ and the case $\mathrm{MMS}_i^k = r$.

A.7 Proof of Lemma 4.4

Similarly to Theorem 3.2, we use an adversary argument assuming that the algorithm only asks queries of the form $\textsc{Eval}(0, x)$ and $\textsc{Cut}(0, \alpha)$.

During the run, there is always a finite set of points $x \in [0, 1]$ for which the algorithm knows the value of $v(0, x)$; we say that such points are *recorded*. Initially only point 0 (which is equivalent to point 1) is recorded. Given a point x, we denote by x_- the closest recorded point counterclockwise, and by x_+ the closest recorded point clockwise. If x itself is recorded, then $x_- = x_+ = x$.

Assume for contradiction that there exists an algorithm as in the theorem statement. We will show how an adversarial agent can answer the queries made by the algorithm. For any integer $t \geq 0$, after t queries are made, the adversary has in mind a valuation function v_t satisfying the following properties:

(i) v_t is compatible with all previously recorded points.

(ii) There is a point x_t such that neither x_t nor $x_t + s$ is recorded, the density between x_t and $x_t + s$ is exactly q/s throughout, and the density elsewhere is strictly greater than q/s throughout.

For the base case $t = 0$, the adversary chooses any $x_0 \notin \{0, 1 - s\}$, and sets the density outside $[x_0, x_0 + s]$ to be constant. Observe that this density is $(1 - q)/(1 - s)$, which is greater than q/s since $q < s$.

For $t \geq 1$, assume that the adversary has in mind the valuation v_{t-1} and the associated point $y := x_{t-1}$. Recall that neither y nor $y + s$ are recorded. If there is no recorded point in the range $(y, y + s)$, the adversary voluntarily records an arbitrary point in that range, and similarly for its complement range $(y + s, y)$. This ensures that y_+ belongs to the interval $(y, (y - s)_-]$. Note that none of the points y_-, y_+, $(y + s)_-$, and $(y + s)_+$ belong to the set $\{y, y + s\}$. When the algorithm makes the t-th query, if both y and $y + s$ would still be unrecorded upon answering according to v_{t-1}, the adversary answers the query according to v_{t-1} and sets $v_t = v_{t-1}$ and $x_t = y$. Else, assume that y would be recorded if the adversary answers the query according to v_{t-1}; the case where $y + s$ would be recorded can be handled analogously. Let z be an arbitrary point in the range (y, y_+) such that $z + s$ belongs to the range $(y + s, (y + s)_+)$. The adversary constructs v_t by making the following changes to v_{t-1}:

(a) Set the density in $[y_-,\ z]$ to be the constant such that the total value of $[y_-,\ y_+]$ is the same in v_t as in v_{t-1}. Note that the density in $[y_-,\ y]$ is smaller and the density in $[y,\ z]$ is larger than in v_{t-1}.

(b) Set the density in $[y + s,\ z + s]$ to be q/s, and the density in $[z + s,\ (y + s)_+]$ to be the constant such that the total value in $[(y + s)_-,\ (y + s)_+]$ is the same in v_t as in v_{t-1}.

See Figure 2 for an illustration of this construction. Observe that in v_{t-1}, the average density in the range $[y_-, y_+]$ is strictly greater than q/s. Since the density in $[z, y_+]$ is set to be exactly q/s in v_t, the constant density in $[y_-, z]$ such that the total value between y_- and y_+ is the same in v_t as in v_{t-1} is strictly greater than q/s. Similarly, the constant density in $[z + s, (y + s)_+]$ in v_t is strictly greater than q/s.

The adversary then answers the query according to v_t. If the query is an eval query, the new recorded point is $y \neq z$. Else, the query is a cut query; in this case, one can check that since $q > 0$ and the point 0 does not belong to the interval (y_-, y_+) (because this interval contains no recorded point), the answer to the query will be a point in the interval (y, z),

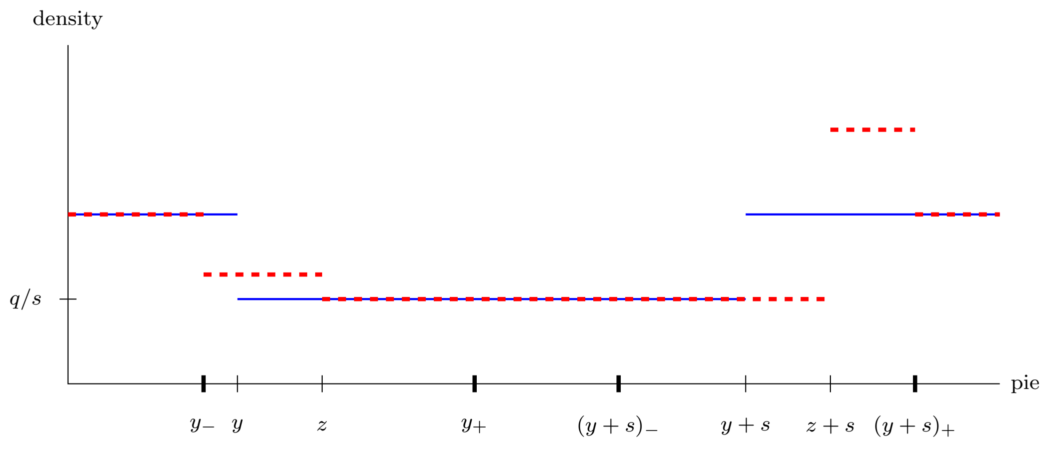


Figure 2: Illustration of the construction in the proof of Theorem 4.4, focusing only on a portion of the pie. The solid blue lines depict the density function of the valuation v_{t-1} and the dashed red lines depict the density function of the valuation v_t. Thicker tickmarks denote recorded points.

which is in particular different from z and $z + s$. This means that v_t satisfies the properties (i) and (ii) with $x_t = z$.

Since the algorithm is finite, it must eventually answer whether there is an interval of length s that the agent values at most q. If it answers no, the adversary reveals that the agent's valuation is v_t, so the value of $[x_t, x_t + s]$ is exactly q. On the other hand, if the algorithm answers yes, the adversary constructs v_{t+1} from v_t by changing the density in $[(x_t)_-, (x_t)_+]$ to be the constant such that the total value in this interval is the same in v_{t+1} as in v_t; by a similar argument as in the previous paragraph, this constant density is strictly greater than q/s. With respect to v_{t+1}, any interval of length s contains portions with density greater than q/s and those with density exactly q/s, with the former taking up a positive amount. This means that such a piece must yield value greater than q to the agent, so by revealing the agent's valuation to be v_{t+1}, the adversary can again falsify the algorithm's answer, completing the proof.

A.8 Proof of Theorem 4.7

Since $\text{MMS}_i^k \le 1/k$ always holds, we only need to decide whether $\text{MMS}_i^k = 1/k$. For the sake of convenience, we modify the queries slightly for pie cutting as follows:

- $\textsc{Eval}_i(x, y)$: Asks agent i to evaluate the interval $[x, y]$ starting from x and going clockwise to y, and return the value $v_i(x, y)$.
- $\textsc{Cut}_i(x, \alpha)$: For $\alpha \le 1$, asks agent i to return the first point y going clockwise from x such that $v_i(x, y) = \alpha$.

It is clear that each of these queries can be implemented using no more than two queries in the original model. If $x > 1$, we identify the point x with the point $x - 1$.

The pseudocode of our algorithm is given as Algorithm 1. We first divide the pie into $\lceil 2/s \rceil$ equal-length intervals, so that the length of each interval is at most $s/2$. For each resulting interval $[x, y]$, if it has value 0, we find the closest point z going clockwise from x such that $v_i(x, z) = 1$; note that z is also the farthest point counterclockwise from x such that $v_i(z, x) = 0$. From point z, we check for a potential partition with min-value $1/k$: we

Algorithm 1 Determining whether $\text{MMS}_i^k = 1/k$ in pie cutting

```
procedure MMS-$1/k$-PIE$(k, s)$
    Divide the pie into $\lceil 2/s \rceil$ equal-length intervals.
    for each resulting interval $[x, y]$ do
        if $\text{EVAL}_i(x, y) = 0$ then
            $z \leftarrow \text{CUT}_i(x, 1)$
            works $\leftarrow$ True
            for $j = 1, 2, \ldots, k$ do
                if $\text{EVAL}_i(z, z + s) \neq 0$ then
                    works $\leftarrow$ False
                $z \leftarrow \text{CUT}_i(z + s, 1/k)$
            if works = True then
                return Yes
    return No
```

insert a separator of size s, jump by value $1/k$, and repeat these steps $k - 1$ more times. If all k separators have value 0, return Yes. Otherwise, if this fails for all candidate intervals $[x, y]$, return No. Since our pie division consists of $O(1/s)$ intervals and we make $O(k)$ queries for each interval, our algorithm uses $O(k/s)$ queries.

Next, we prove the correctness of the algorithm. If the algorithm returns Yes, then the k separators together with the pieces in between form a partition with min-value $1/k$, so $\text{MMS}_i^k = 1/k$.

For the converse direction, suppose that $\text{MMS}_i^k = 1/k$, so there exists a partition with min-value $1/k$. We can assume without loss of generality that each separator in our partition cannot be extended in either direction without reducing the value of the adjacent parts, i.e., each separator is an inclusion-maximal interval of value 0 (in particular, we allow separators to have length greater than s). Since the length of each separator in this partition is at least s, there exists an interval in our initial pie division that is contained in one of the separators; let $[x, y]$ be one such interval. It suffices to show that the algorithm returns Yes when starting with the interval $[x, y]$ in the for-loop.

Since $[x, y]$ is contained in a separator, we have $v_i(x, y) = 0$. Denote by S_1 the separator containing $[x, y]$. Let P_1 be the adjacent piece of the partition going clockwise, and denote the following separators and pieces by $S_2, P_2, \ldots, S_k, P_k$. Considering all pieces in the clockwise direction, we find that z coincides with the starting point of S_1. Since S_1 has length at least s, the separator that the algorithm inserts has value 0; moreover, when the algorithm jumps by value $1/k$, it will reach exactly the endpoint of P_1, which is also the starting point of S_2. Repeating this argument, we conclude that all k separators that the algorithm inserts indeed have value 0, and the algorithm returns Yes, as claimed.

A.9 Proof of Theorem 4.8

Assume for contradiction that such an algorithm exists, and let $0 < s < \frac{1}{4c}$. We will show how an adversary can answer the queries of the algorithm in such a way that after at most c queries, there exists a piecewise constant valuation function consistent with the answers for which $\text{MMS}_i^2 \geq 1/2$, but also one for which $\text{MMS}_i^2 < 1/2$. This is sufficient to obtain the desired contradiction. Since it always holds that $\text{MMS}_i^2 \leq 1/2$, the first case is equivalent to $\text{MMS}_i^2 = 1/2$.

The adversary records any point that appears in a query or an answer to it, and answers queries as if the valuation is uniform. Suppose that at most c queries have been answered in this manner. Each query and its answer increase the number of recorded points by at

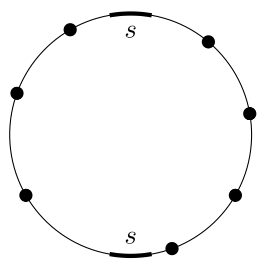


Figure 3: Illustration for the proof of Theorem 4.8, with a pie and its recorded points. The two highlighted intervals are "good" intervals.

most two, so there are at most $2c$ recorded points. Call an interval of length s "good" if neither it nor the diametrically opposite interval of length s contains any recorded point inside or on its border (see Figure 3). For each recorded point, the set of points that it rules out as potential centers of a good interval is a union of two intervals of length s; since $2c \times 2s < 1$, a good interval exists. Choose one such interval along with the diametrically opposite interval, and if necessary, record additional points so that there is at least one recorded point on both pieces of the pie between the two intervals. Consider the following two possibilities:

Possibility 1: The entire valuation function is uniform. Since $s > 0$, we have $\text{MMS}_i^2 < 1/2$ in this case.

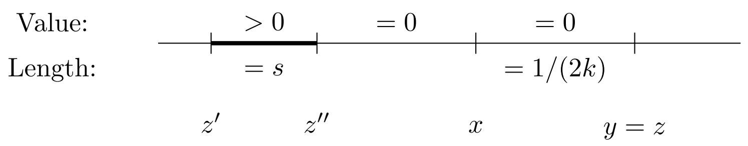
Possibility 2: Consider a piece between any two consecutive recorded points. If the piece does not contain one of the two intervals, the adversary simply distributes the value uniformly across the piece. Else, the adversary "shifts" the value away from the interval in the following manner: Divide the interval into two subintervals of length $s/2$, and for each subinterval, move its value to the adjacent part of the same piece outside the interval. This can be done so that the resulting valuation is piecewise constant. Since the two intervals of length s serve as separators for a partition with min-value $1/2$, we have $\text{MMS}_i^2 = 1/2$.

Therefore, a finite algorithm using at most c queries cannot distinguish between the case $\text{MMS}_i^2 < 1/2$ and the case $\text{MMS}_i^2 \geq 1/2$.

A.10 Proof of Theorem 4.9

We work with the same Cut and Eval queries as in Theorem 4.7, and use $[a, b]$ and $[a, b)$ to refer to the closed and half-open interval going clockwise from a to b, respectively. Addition is taken modulo 1.

The pseudocode of our algorithm is given as Algorithm 2. We first divide the pie into $2k$ equal-length intervals. If all of these intervals yield nonzero value, we return Yes. Else, we consider one of the intervals with value 0, say $[x, y]$. Starting from y, we find the farthest point z'' counterclockwise such that the interval $[z'', y]$ has value 0—this is done by making a cut query of value 1 clockwise from y—and insert a separator $[z', z'']$ of length s:



We then repeat this process $k - 2$ more times starting from z'. If we cover the entire pie and reach point y at any stage during the process, we return No. Else, we return Yes. The first step where we check whether all $2k$ intervals are of positive value requires $O(k)$ eval

Algorithm 2 Determining whether $\text{MMS}_i^k > 0$ in pie cutting when $s \le \frac{1}{2k}$

```
1: procedure MMS-0-PIE($k, s$)
2:     Divide the pie into $2k$ equal-length intervals.
3:     if all $2k$ intervals have a positive value for agent $i$ then
4:         return Yes
5:     $[x, y] \leftarrow$ one of the intervals with value 0
6:     $z \leftarrow y$
7:     for $j = 1, 2, \ldots, k-1$ do
8:         $z'' \leftarrow \text{CUT}_i(z, 1)$
9:         if the interval $[z'', z)$ contains $y$ then
10:            return No
11:        $z' \leftarrow z'' - s$
12:        if the interval $[z', z'')$ contains $y$ then
13:            return No
14:        $z \leftarrow z'$
15:    if $\text{EVAL}(y, z) = 0$ then
16:        return No
17:    return Yes
```

queries, while the remaining steps take $O(k)$ cut queries, so the total number of queries is $O(k)$.

Next, we prove the correctness of the algorithm. If all $2k$ intervals of length $\frac{1}{2k}$ are of positive value, then since $s \le \frac{1}{2k}$, the partition with all odd-numbered intervals as separators has a positive min-value, and our algorithm correctly returns Yes. Assume now that there exists a zero-valued interval $[x, y]$, and consider the for-loops. By definition of the cut query, the intervals $[z'', z]$ (some of which may be empty) have zero value. Hence, if the algorithm returns No, the entire value of the pie is covered by the (at most) $k-1$ separators $[z', z'']$. When this is the case, we indeed have $\text{MMS}_i^k = 0$: for any s-separated partition of the pie into k parts, at most one part can overlap each interval $[z', z'']$, so at least one part necessarily has zero value.

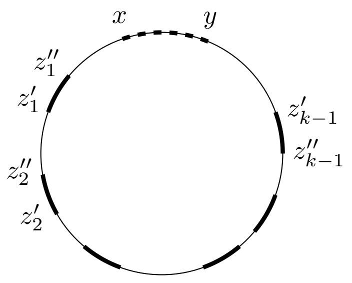


Figure 4: Illustration for the proof of Theorem 4.9. The interval $[x, y]$ is one of the original $2k$ equal-length intervals and has value 0. Each interval $[z'_i, z''_i]$ is a separator of length s inserted by the algorithm.

Finally, suppose the algorithm returns Yes in the last line of the algorithm, and call the $k-1$ separators inserted by the algorithm $[z'_1, z''_1], \ldots, [z'_{k-1}, z''_{k-1}]$ in this order (see Figure 4). The interval $[y, z'_{k-1}]$ has a positive value by line 15 of the algorithm. Therefore, we can slightly move the separator $[z'_{k-1}, z''_{k-1}]$ towards y, so that $[y, z'_{k-1}]$ still retains a positive value. By definition of z''_{k-1}, now the interval $[z'_{k-2}, z''_{k-1}]$ has a positive value. Applying the same argument $k-1$ times, we see that the entire set of separators can be slightly moved to yield a k-partition with a positive min-value and $k-1$ separators of length s. The interval $[x, y]$, which has length $\frac{1}{2k} \geq s$, then serves as the k-th separator of a partition with a positive min-value.

A.11 Proof of Theorem 4.11

Mark points on the pie so that for any two adjacent marks the value of the piece between them is at most $\varepsilon/2$; let M be the set of marked points. Let r denote the highest min-value among all s-separated partitions such that both endpoints of each of the k pieces are in M. We first claim that $r \geq \text{MMS}_i^k - \varepsilon$. Indeed, consider a maximin partition, and shrink each of the k pieces by moving each endpoint until it coincides with a marked point. The resulting partition is still s-separated, and the value lost by each piece is at most $\varepsilon/2 + \varepsilon/2 = \varepsilon$. Since the min-value of the original partition is MMS_i^k, the min-value of the new one is at least $\text{MMS}_i^k - \varepsilon$.

Our algorithm starts by marking points as above. For each interval $[x, y]$ where $x, y \in M$, we attempt to construct an s-separated partition with min-value $v_i(x, y)$ that has $[x, y]$ as one of its pieces and such that the endpoints of all k pieces are in M. The construction proceeds in a greedy fashion: starting with $[x, y]$ and going clockwise, we add the smallest separator of length at least s that ends at another marked point (say, x'), create the next piece by finding the smallest $y' \in M$ such that $v_i(x', y') \geq v_i(x, y)$, add another separator of length at least s, and so on. If we can add n separators without overlapping with the original interval $[x, y]$, the construction succeeds. We return the highest value $v_i(x, y)$ such that the construction succeeds, along with the corresponding partition.

Note that the algorithm only needs $O(1/\varepsilon)$ queries in order to mark points in the first step; the additional steps do not require further queries. To see its correctness, consider a partition with min-value r defined in the first paragraph, and let $[x, y]$ be a piece in this partition with value exactly r. The algorithm succeeds when it starts with $[x, y]$: this follows from a greedy argument similar to that in Theorem 3.5. Hence the value returned by the algorithm is at least $r \geq \text{MMS}_i^k - \varepsilon$. Moreover, the algorithm necessarily outputs the min-value of an s-separated partition, which is at most MMS_i^k by definition. This concludes the proof.

B CutRight Query

We show that the $\textsc{CutRight}_i(x, \alpha)$ query, which returns the rightmost point y for which $v(x, y) = \alpha$, cannot be implemented using finitely many queries in the standard Robertson–Webb model. This query has been used by Cechlárová and Pillárová [2012], where it was called a "reverse cut" query.

Theorem B.1. *For any agent i and real number $\alpha \in (0, 1)$, let $\textsc{CutRight}_i(0, \alpha)$ be the largest $x \in (0, 1)$ for which $v_i(0, x) = \alpha$. There is no algorithm that computes $\textsc{CutRight}_i(0, \alpha)$ by asking agent i a finite number of Robertson–Webb queries.*

Proof. Suppose for contradiction that $\textsc{CutRight}_i(0, \alpha)$ can be computed using finitely many Robertson–Webb queries. An adversary can answer all queries as if v_i is uniform, i.e., for any two consecutive recorded points on the cake, if the piece between them has length t, then it also has value t. After any finite number of queries, it is possible that the entire valuation is uniform, in which case the algorithm should answer α. But it is also possible that, for some small $\varepsilon > 0$, it holds that $v_i(\alpha, \alpha + \varepsilon) = 0$, where $\alpha + \varepsilon$ is smaller than the next recorded point (i.e., the next point involved in any query or answer). In this case, the algorithm should answer at least $\alpha + \varepsilon$. □