Fair-cake cutting in Computer Science

Hăbeanu Flavius-Ștefan
Iulia Rusu

Summarised: May 2025

Outline

1. Justification
    - General Context
    - Models and Fundamental Assumptions
    - Interpretation in Computer Science
2. Main points
    - Justice Requirements
    - Algorithms and Methods for Fair Division
    - Application of Fair Division Algorithms in the Theory of Computation
3. Conclusion

1 Justification

1.1 General Context

The equitable division of resources has represented a fundamental challenge since ancient times. The "cake" is only a metaphor. Fair cake-cutting procedures can be used to divide various types of resources, such as land estates, advertising space, or broadcast time. Therefore, this issue extends to any divisible resource, from the metaphorical cutting of a cake (which inspired the project's title, Fair Cake-Cutting) to resource allocation problems encountered in computer science, such as task scheduling on processors, memory management, or access time distribution within computing systems.

In the field of computer science, fairness-related problems are frequently encountered in domains such as distributed systems, artificial intelligence, algorithmic game theory, and increasingly in emerging areas like blockchain and cloud computing. Developing an effective solution requires more than just fairness. It must also be algorithmically computable, meaning that it must conform to the computational constraints of the systems in which it is applied.

The primary objective of this project is to provide a clear and structured overview of the fair cake-cutting problem. This includes an exposition of the core fairness criteria, the principal algorithms designed to address them, and an exploration of the connections between fair division theory and real-world computational challenges.

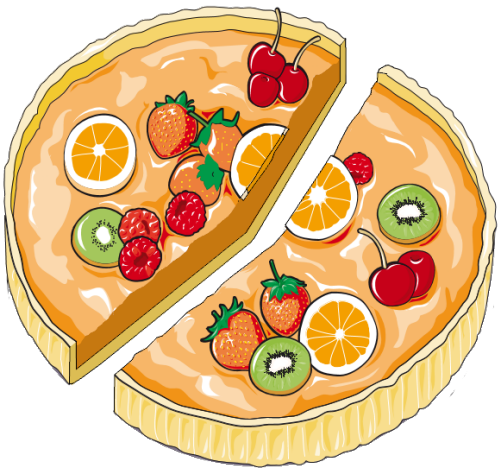

Figure 1: Cake-cutting in 2 slices

1.2 Models and Fundamental Assumptions

The fair cake-cutting problem can be mathematically formulated as follows:

We are given a divisible good (the "cake"), modeled as a continuous interval $[0, 1]$, which must be divided among n agents. Each agent possesses a subjective

evaluation of the cake, captured by a utility function v_i, which assigns a value to each subinterval.

The Cake

Modeled as the real interval $[0, 1]$, representing a continuous resource, such as time, data, or memory.

Agents

Let $A_1, A_2, \ldots, A_n$ be the agents, each associated with a utility function

$$v_i : [0, 1] \to \mathbb{R}^+.$$

These utility functions are typically assumed to satisfy the following properties:

- **Normalization:**
$$v_i([0, 1]) = 1$$
- **Additivity:** For disjoint intervals $[a, b]$ and $[c, d]$,
$$v_i([a, b] \cup [c, d]) = v_i([a, b]) + v_i([c, d])$$
- **Continuity:** There are no abrupt changes or infinite jumps in value — an essential condition to ensure divisibility.

Allocation

An allocation is defined as a partition of the interval $[0, 1]$ into n disjoint subintervals $X_1, X_2, \ldots, X_n \subseteq [0, 1]$, such that:

$$\bigcup_{i=1}^{n} X_i = [0, 1] \quad \text{and} \quad X_i \cap X_j = \emptyset \quad \text{for } i \neq j,$$

where each X_i represents the portion allocated to agent A_i.

1.3 Interpretation in Computer Science

In computational contexts, the "cake" metaphor extends to various divisible resources, such as:

- **CPU Time**, as in process scheduling.
- **Disk Space** when shared among multiple users.
- **Access Rights** to a resource in a distributed system.

The agents' utility functions in these settings may reflect:

- The urgency of a computational process.
- The resource demands of a software application.
- The priority level of a user in a multi-agent or multi-user system.

2 Main points

2.1 Justice Requirments

In order for the cake to be divided fairly, certain fairness criteria must be satisfied. These are mathematically defined based on the agents' preferences and, in computational contexts, correspond to desirable properties of allocation algorithms.

Proportionality

An allocation is **proportional** if each agent receives at least their fair share — that is, at least $\frac{1}{n}$ of the total value according to their own utility function:

$$v_i(X_i) \geq \frac{1}{n}, \quad \text{for all } i.$$

In practical terms, this means that in a system with n processes requesting resources, each process receives at least one-n^{th} of the total resource, as measured by its own internal evaluation or priority model.

Example: If three agents are dividing a cake they each value differently, proportionality ensures that each agent gets a piece they value at least one-third of the total, even if the pieces are not equal in size or shape.

Limitation: Proportionality guarantees a minimum level of satisfaction, but does not account for whether agents feel they received less than others.

Envy-Freeness

An allocation is **envy-free** if no agent prefers another agent's portion to their own:

$$v_i(X_i) \geq v_i(X_j), \quad \text{for all } i, j.$$

In multi-agent systems, this guarantees that each agent believes their own allocation is at least as good as anyone else's. This is a stronger condition than proportionality, as it ensures fairness not just in isolation, but in comparison.

Example: In a file-sharing system, if two users are given download slots, an envy-free allocation ensures no user believes the other's slot is better than their own.

Key insight: Every envy-free allocation is also proportional, but not every proportional allocation is envy-free.

Pareto Optimality (Efficiency)

An allocation is **Pareto-optimal** if no other allocation can make at least one agent better off without making someone else worse off:

$$\nexists (X'_1, \ldots, X'_n) \text{ such that } \forall i : v_i(X'_i) \geq v_i(X_i) \text{ and } \exists j : v_j(X'_j) > v_j(X_j).$$

In resource allocation problems, such as distributed systems, this means that we cannot improve one agent's satisfaction without harming another's.

Example: If one agent gets a piece of cake they value at 40, and another gets a piece worth 60, but we could reallocate to 50 and 60, the original allocation is not Pareto-optimal.

Strategy-Proofness (Optional)

An allocation algorithm is **strategy-proof** if agents cannot benefit from misrepresenting their preferences.

In automated contexts (e.g., bandwidth allocation in networks), this property ensures that agents cannot "game the system" to obtain more resources by lying about their true utility functions.

Example: In automated bandwidth allocation, a node may exaggerate its needs to gain more resources. A strategy-proof algorithm makes such manipulation useless or even harmful to the agent.

Significance: This property is especially important in decentralized systems, where incentives for strategic behavior are high.

Interrelationships Between Fairness Criteria

- Any envy-free allocation is also proportional, but the converse does not always hold.
- Pareto efficiency may conflict with envy-freeness. Maximizing utility does not always guarantee perceived fairness.
- Not all combinations of fairness properties are simultaneously achievable. In many cases, trade-offs are necessary.

Fairness Criterion	Individual Fairness	Relational Fairness	Efficiency	Strategy Resistance
Proportionality	Yes, each agent receives at least their fair share	No, doesn't ensure fairness compared to others	No, not necessarily efficient	No, agents may benefit from lying
Envy-Freeness	Yes, each agent feels content with their share	Yes, no one prefers another's share	No, may not be optimal globally	No, can still be manipulated
Pareto Optimality	No, individual satisfaction not guaranteed	No, doesn't prevent perceived inequality	Yes, globally efficient	No, incentives to misreport may remain
Strategy-Proofness	No, may not guarantee a minimum fair share	No, may still be seen as unfair compared to others	Yes, often aims for global utility	Yes, truth-telling is the best strategy

Figure 2: Fairness Comparison

2.2 Algorithms and Methods for Fair Division

2.2.1 Cut and Choose (Two Agents)

The **Cut and Choose** method is one of the simplest and most well-known fair division algorithms, designed for scenarios involving two agents:

1. The first agent divides the cake into two portions that are equal according to their own utility function.
2. The second agent then selects the piece they prefer.

This mechanism guarantees the following fairness properties:

- **Envy-freeness**: Neither agent prefers the other's portion.
- **Proportionality**: Each agent receives at least half of the total value according to their own evaluation.

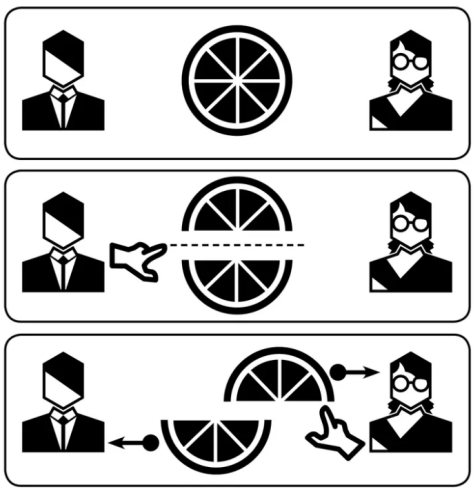

Figure 3: Cut and Choose

Application in Computer Science

The *Cut and Choose* protocol can be applied to the division of computational resources between two processes with conflicting interests.

For example, it can be used to fairly allocate CPU time between two threads, where each thread has a different evaluation of what constitutes a valuable time slot.

2.2.2 Steinhaus' Algorithm (n Agents)

Steinhaus' algorithm generalizes the classic *Cut and Choose* method to scenarios involving more than two agents.

Procedure

Agents take turns dividing the cake, and subsequent agents choose portions based on their preferences. The process ensures that each agent receives at least a fair share according to their own utility function.

Fairness Properties

- **Proportionality**: Guaranteed, as each agent receives at least $\frac{1}{n}$ of the cake in their own valuation.

- **Envy-freeness**: Not guaranteed, as an agent may still prefer another agent's portion.
- **Practicality**: The algorithm is most effective when the number of agents is relatively small, due to its sequential and preference-dependent nature.

Complexity

The algorithm proceeds by **dividing agents one by one**. At each step, **only one cut and one choice** are made. The cake is split recursively, reducing the number of agents by one at each step.Total number of steps is proportional to the number of agents n. Hence, **each of the n agents is handled once**, resulting in a linear O(n) time complexity.

Applications

Steinhaus' method is suitable for small-scale resource allocation problems in computer science, such as:

- Fair task assignment among a small group of agents.
- Division of limited bandwidth or memory blocks among a few competing processes.

2.2.3 Last Diminisher Algorithm

The **Last Diminisher** protocol is a classic fair division algorithm that ensures proportionality among any number of agents.

Procedure

1. The first agent cuts a piece they believe to be proportional (i.e., worth exactly $\frac{1}{n}$ of the total cake).
2. Each subsequent agent inspects the piece and may trim it if they believe it exceeds their fair share.
3. The last agent to modify (diminish) the piece receives it.
4. The process then repeats among the remaining agents, using the leftover cake.

Fairness Properties

- **Proportionality**: Guaranteed for any number of agents n.
- **Envy-Freeness**: Not guaranteed, as agents may still prefer others' portions.
- **Correctness**: The method is fair under the proportionality condition.
- **Efficiency**: Can be inefficient in practice, with linear time complexity in n for each allocation round, leading to significant overhead as n increases.

Complexity

Setup

- There are n agents.
- In each round, one piece of the cake is cut and allocated
- The process repeats for the remaining n-1, then n-2, and so on

The total number of evaluations is:

$$n + (n-1) + (n-2) + \cdots + 1 = \frac{n(n+1)}{2} = O(n^2)$$

Therefore, the time complexity of the Last Diminisher algorithm is $O(n^2)$ because the number of inspections and potential cuts grows quadratically with the number of agents.

Applications

This algorithm is suitable for systems where strict fairness is required and the number of agents is moderate, such as:

- Shared memory access among multiple users.
- Time-slot allocation in cooperative scheduling systems.

2.2.4 Selfridge–Conway Algorithm (Three Agents)

The **Selfridge–Conway algorithm** is a more intricate method designed specifically for three agents, ensuring stronger fairness guarantees than proportionality alone.

Procedure

It uses a combination of cutting, trimming, and redistribution steps to ensure that each agent receives a portion they do not envy, even after considering others' portions.

Fairness Properties

- **Envy-Freeness**: Guaranteed, as no agent prefers another's share.
- **Proportionality**: Also guaranteed.
- **Complexity**: Higher than simpler methods, the algorithm involves multiple rounds and subprocedures, increasing its computational and logical complexity.

Detailed Complexity Analysis

- The algorithm involves multiple **cutting and trimming operations**, where agents evaluate and possibly adjust pieces based on their utility functions.
- At least **two or three rounds of actions** take place, involving careful comparison of portions and reallocation steps.
- Unlike Last Diminisher, the number of agents n is fixed at 3, so the complexity is **constant time O(1)** with respect to the number of agents, but the steps are logically complex.
- However, in terms of **computational effort per agent**, the algorithm requires more intricate preference checks and potential recalculations compared to simpler proportional algorithms.
- The **logical complexity** arises from the conditional branching and the need to track trimming and reallocations, making implementation more involved.
- For practical purposes in multi-agent systems with exactly three agents, the increased complexity is justified by the stronger fairness guarantees, especially envy-freeness.

Applications

The Selfridge–Conway protocol is particularly relevant in collaborative computing environments involving three users or agents, such as:

- Cloud-based resource sharing among multiple tenants.
- Collaborative decision-making systems in distributed AI or negotiation scenarios.

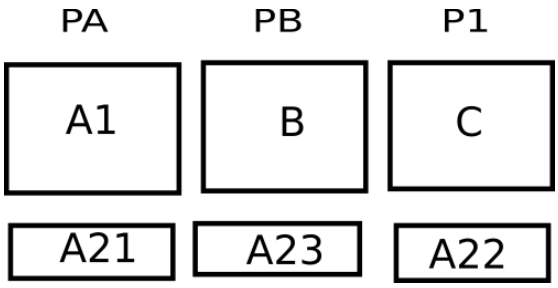

Figure 4: Selfridge–Conway Algorithm

2.2.5 Modern Computational Algorithms

Contemporary approaches to fair division employ formal frameworks to define the computational capabilities of algorithms. One widely used framework is the **Robertson–Webb model**, which specifies the types of queries an algorithm is allowed to make when interacting with agents.

Robertson–Webb Model

This model defines two basic types of oracle queries:

- `Eval`(x, y): "What is the value of the interval $[x, y]$ according to your utility function?"
- `Cut`(x, r): "What is the point y such that the interval $[x, y]$ has value r?"

By abstracting the interaction between the algorithm and agents' preferences, this model allows for the design and analysis of algorithms with formal guarantees.

Usage

- **Abstraction:** Agents are treated as black boxes — the algorithm doesn't need to know the full utility function, only how they respond to queries.
- **Universality:** Many known algorithms (e.g., Last Diminisher, Dubins–Spanier, Even–Paz) can be formalized within this model.
- **Comparison Tool:** The model allows complexity comparison of algorithms by counting the number of queries they require.
- **Precision:** Helps define what it means for an algorithm to be efficient or exact under partial information.

Approximation Algorithms

Given the computational hardness of guaranteeing envy-freeness or Pareto optimality for arbitrary numbers of agents, modern algorithms often focus on approximate solutions. These algorithms aim to:

- Minimize envy below a specified threshold (ε-envy-freeness),
- Achieve approximate efficiency while maintaining a degree of fairness,
- Run in polynomial time, making them scalable for real-world applications.

Robertson-Webb based algorithms

Dubins-Spanier Algorithm

The **Dubins–Spanier** algorithm is a classic continuous procedure for fair division that guarantees **proportionality** for any number of agents n. It belongs to the class of **moving-knife algorithms**, meaning it is conceptual and involves a continuous movement across the resource (e.g., cake).

Procedure (for n agents)

1. An imaginary knife moves continuously from left to right across the cake.
2. Each agent watches the knife and calls "Stop!" when the portion to the left of the knife equals 1/n of the total value according to their own utility function.

3. The first agent to call "Stop!" receives the piece and exits the process.
4. The procedure is repeated recursively with the remaining cake and n-1 agents.

Fairness Properties

- **Proportionality:** Yes, each agent receives a piece they value as at least 1/n
- **Envy-Freeness**: No, an agent might prefer another's portion
- **Pareto Efficiency:** Not always, it does not guarantee utility maximization
- **Strategy-Proofness:** No, agents may gain by misrepresenting their preferences.

In the Robertson–Webb model (used to analyze computational fair division algorithms):

- Each agent may be asked **Eval(x, y)** or **Cut(x, r)** queries.
- In each round, up to n queries may be needed.
- There are n rounds (as one agent is removed each time).

Therefore, the total query complexity is $O(n^2)$.

Even–Paz algorithm

The **Even–Paz algorithm** is a proportional fair division method designed to efficiently divide a divisible good among n agents. It guarantees that each agent receives **at least 1/n of the total value**, according to their own preferences. Unlike simple cut-and-choose methods, it scales well for a large number of agents and has relatively low computational complexity.

Conceptual Description

1. **Divide-and-conquer strategy:** The algorithm recursively divides both the cake and the group of agents into two parts.
2. **Cut phase:** Each agent is asked to make a cut that, in their view, separates the cake into a "smaller" and "larger" portion (e.g., the first third and the remaining two-thirds). This cut is based on what they consider to be a fair share.
3. **Median selection:** From all the cuts, the algorithm chooses a median cut, one that balances the valuations.
4. **Partition of agents:** Agents are split into two groups: Those who believe the piece before the median cut is at least their fair share, and the rest, who think the piece after the cut is more valuable to them.
5. **Recursive call:** The same process is applied to each subgroup of agents and their respective piece of cake, continuing until each agent gets a portion.

Complexity analysis

- At each recursion level:
  - You perform n cut queries → each takes $O(log 1/\varepsilon)$ for precision ε
  - You compute the median of n points → $O(n)$
- The depth of the recursion tree is $O(logn)$, because the agents are split roughly in half at each step.

The Even–Paz algorithm uses only Cut(i, r) queries from the Robertson–Webb model: **Cut(i, r)** → Asks agent i to identify the smallest piece (starting from a given point) that has value exactly r, according to their preferences.

This makes the algorithm efficient and simple to implement in computational environments, since it doesn't need to evaluate the agents' preferences directly. It only interacts through well-defined queries.

Applications in Computer Science

Modern fair division algorithms have significant implications in computational domains, particularly in resource allocation problems, such as:

- **Fair Scheduling**: Equitable allocation of CPU time or memory among competing processes.
- **Distributed Systems**: Dynamic resource distribution among agents with heterogeneous preferences.
- **Task Allocation Platforms**: Algorithms for fairly assigning tasks to autonomous agents, workers, or services in multi-agent systems.

These applications require algorithms that are not only fair but also efficient, scalable, and implementable in real-time systems.

3 Application of Fair Division Algorithms in CS112

Fair division problems, such as cake-cutting, may seem far from the Theory of Computation at first glance. However, **the logical structure, computational models, and algorithmic reasoning** behind these protocols have deep connections to the subject.

1. **Formal Modeling of Preferences:** In cake-cutting, agents' preferences are modeled mathematically through utility functions. These can be expressed using formal languages that define valid input (e.g., value intervals) and operations (e.g., cut, eval). Such a modeling approach mirrors how languages describe computation inputs in automata theory.
   - For instance, preferences can be encoded using a regular grammar over intervals, and an automaton can simulate evaluation steps.
2. **Automata as Allocation Agents:** In computational implementations, each agent in a fair division protocol could be modeled as a finite automaton or a Turing machine that processes inputs (cake intervals) and outputs decisions (accept, cut, trim). The interactions between agents and the protocol simulate automata communicating via inputs/outputs.

- Example: In the Even–Paz algorithm, each agent acts like a DFA that takes a value r and returns a cut point. This is conceptually similar to how transducers work.

3. **Decision Processes as Computational Trees:** The recursive structure of algorithms like Even–Paz or Dubins–Spanier resembles computation trees, which are central in automata-based parsing and recursive language recognition (e.g., LL or LR parsers). Each step represents a node in a decision tree, with branches depending on cuts/evaluations.

    - Example: The process of narrowing down the cake intervals is structurally similar to parsing a recursive language by binary division of a string.

4. **Algorithmic Complexity Analysis:** Formal language theory deals heavily with complexity classes, such as P, NP, etc. Many fair division algorithms have well-defined complexity bounds (e.g., $O(nlogn)$), and studying these bounds involves the same tools as analyzing state transitions or grammar parsing time.

    - Example: Analyzing the Robertson–Webb model is similar to studying how many transitions an automaton must perform to accept a string.

5. **Symbolic Representation:** Protocols like Selfridge–Conway or Last Diminisher can be written as symbolic transition systems, much like automata or pushdown automata, especially when formalizing each round's logic or agent decision-making.

4 Conclusion

Fair division problems, such as the fair cake-cutting model, provide a powerful framework for designing algorithms that allocate resources in a just and principled way. Algorithms like *Cut and Choose*, *Last Diminisher*, and *Selfridge–Conway* illustrate different strategies tailored to the number of agents involved and the fairness criteria pursued.

Beyond the metaphor of cake-cutting, these ideas have practical applications in computer science, ranging from CPU scheduling in operating systems to task distribution in networks of intelligent agents. Fair division models enable the development of algorithms that are not only equitable, but also transparent and strategy-resistant, all of which are crucial for building robust and trustworthy computational systems.

Ongoing research continues to explore the extension of these algorithms to dynamic, discrete, or uncertain environments, such as distributed systems and online markets. The integration of computational fairness into software design is becoming increasingly important in the broader context of algorithmic ethics and automated decision-making.

References

[1] Wikipedia contributors. *Envy-free cake cutting.*
https://en.wikipedia.org/wiki/Envy-free_cake_cutting

[2] Reddit. *Fair cake cutting — is it doable?*, r/askmath.
https://www.reddit.com/r/askmath/comments/8an63w/fair_cake_cutting_is_it_doable/

[3] Paul K. Stockmeyer. *The Mathematics of Cake Cutting*, Scientific American.
https://www.scientificamerican.com/article/the-mathematics-of-cake-cutting/

[4] Wikipedia contributors. *Divide and choose.*
https://en.wikipedia.org/wiki/Divide_and_choose

[5] Wikipedia contributors. *Selfridge–Conway procedure.*
https://en.wikipedia.org/wiki/Selfridge%E2%80%93Conway_procedure

[6] Subhash Suri. *Fair Division and Cake Cutting.*
https://sites.cs.ucsb.edu/~suri/ccs130a/Fairness.pdf

[7] Hiro Ito and Takahiro Ueda. *How to Solve the Cake-Cutting Problem in Sublinear Time.*
8th International Conference on Fun with Algorithms (FUN 2016).
https://drops.dagstuhl.de/storage/00lipics/lipics-vol049-fun2016/LIPIcs.FUN.2016.21/LIPIcs.FUN.2016.21.pdf