This is the Project Title

Your Name

Master of Science

School of Informatics

University of Edinburgh

2024

Abstract

This skeleton demonstrates how to use the `infthesis` style for MSc dissertations in the School of Informatics. It also emphasises the page limit and associated style restrictions for Informatics dissertations with course code `INFR11077`. If your degree has a different project course code, then it is likely to have different formatting rules. The file `skeleton.tex` generates this document and should be used as a starting point for your thesis. Replace this abstract text with a concise summary of your report.

Research Ethics Approval

Instructions: *Agree with your supervisor which statement you need to include. Then delete the statement that you are not using, and the instructions in italics.*

Either complete and include this statement:

This project obtained approval from the Informatics Research Ethics committee.
Ethics application number: ???
Date when approval was obtained: YYYY-MM-DD
[If the project required human participants, edit as appropriate, otherwise delete:]
The participants' information sheet and a consent form are included in the appendix.

Or include this statement:

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Your Name*)

Acknowledgements

Any acknowledgements go here.

Table of Contents

Chapter 1

Introduction

The preliminary material of your report should contain the following:

- The title page.
- An abstract page.
- Declaration of ethics and own work.
- Optionally, an acknowledgments page.
- The table of contents.

As in this example `skeleton.tex`, the above material should be included between:

```
\begin{preliminary}
    ...
\end{preliminary}
```

This style file uses Roman numeral page numbers for the preliminary material.

The main content of the dissertation, starting with the first chapter, starts on page 1. ***The main content must not go beyond page 40.***

The report then contains a bibliography and any appendices which may go beyond page 40. The appendices are only for any important supporting material to go on record. However, you cannot assume markers of dissertations will read them.

You may not change the dissertation format (e.g., reduce the font size, change the margins, or reduce the line spacing from the default 1.5 spacing). Be careful if you copy-paste packages into your document preamble from elsewhere. Some LaTeX packages, such as `fullpage` or `savetrees`, change the margins of your document. Do not include them!

Over-length or incorrectly-formatted dissertations will not be accepted, and you would have to modify your dissertation and resubmit. You cannot assume we will check your submission before the final deadline, and if it requires resubmission after the deadline to conform to the page and style requirements, you will be subject to the usual late penalties based on your final submission time.

1.1 Using Sections

Divide your chapters into sub-parts as appropriate.

1.2 Citations

Citations (such as `BibTeX`. For more advanced usage, we recommend using the `natbib` package or the newer `biblatex` system.

These examples use a numerical citation style. You may use any consistent reference style that you prefer, including "(Author, Year)" citations.

Chapter 2

The Brânzei-Nisan algorithm

2.1 Input preprocessing

2.2 Eval and Cut Queries

The main tools to be implemented before beginning work on this algorithm are the eval and cut queries. These act much the same as the RW equivalents, but they must be implemented in a discretised manner in order to be used in computer algorithms. Brânzei-Nisan's solution to this discretisation, beginning with eval queries, is as follows

- first, ensure the value functions are strictly hungry for all agents i (that is $v_i([a,b]) > 0 \forall [a,b] \subseteq [0,1]$) via the following preprocessing:

$$v_i'([a,b]) = (1-\varepsilon/2)\cdot v_i([a,b]) + \varepsilon/2$$

- Then, the valuation functions are approximated via piecewise-constant functions on an ε-grid $\{0, \varepsilon, 2\varepsilon, \ldots, 1-\varepsilon, 1\}$ such that

  - if x is divisible by ε:

$$v_i''([0,x]) = \lceil v_i'([0,x]) \rceil_\varepsilon \tag{2.1}$$

    (Essentially, the value up to the nearest integer multiple of ε). By additivity, we can then find $v_i''([a,b])$, given that both a and b are divisible by ε, by subtraction

$$v_i''([a,b]) = v_i''([0,b]) - v_i''([0,a]). \tag{2.2}$$

– if x is not divisible by ε, find k such that $k\varepsilon < x < (k+1)\varepsilon$. We know by additivity that

$$v_i''([0,x]) = v_i''([0,k\varepsilon]) + v_i''([k\varepsilon,x]). \tag{2.3}$$

As $k\varepsilon$ is divisible by ε, $v_i''([0,k\varepsilon])$ can be found via (2.1). For $v_i''([k\varepsilon,x])$, interpolation is used such that

$$v_i''([k\varepsilon,x]) = \frac{x-k\varepsilon}{\varepsilon}\, v_i''([k\varepsilon,(k+1)\varepsilon]). \tag{2.4}$$

Using these functions, $v_i''([a,b])$ is defined for any $[a,b] \subseteq [0,1]$[1].

When implementing this modified valuation function in the website, there is no necessity to change the valuation functions retrieved from the user input as the modifications can be performed as the algorithm is running.

Cut queries are then implemented as a binary search of value queries, such that if you pass in an initial cut position a, value α, and whether the initial cut is an end cut or start cut of a slice, it will run a binary search of value queries until both the upper and lower bounds of the returned cut lie within the same interval of the epsilon grid. At this point, it is unnecessary to continue the binary search as we now have all the necessary information to find the exact cut b (according to the modified valuation function). The details of how to find the exact cut position from this point will be more thoroughly fleshed out in the four-agent algorithm, as this is where it has a more prominent effect on runtime when compared to continuing the binary search. Details for finding exact cut positions in this case will be provided in the appendix.

$$\begin{aligned}
v_i''([0,b]) &= v_i''([a,b]) + v_i''([0,a]) && \text{by (2.2)}\\
v_i''([0,b]) &= \alpha + v([0,a])\\
v_i''([0,k\varepsilon]) + v_i''([k\varepsilon,b]) &= \alpha + v([0,a]) && \text{by (2.3)}\\
v_i''([k\varepsilon,b]) &= \alpha + v([0,a]) - v_i''([0,k\varepsilon])\\
\frac{b-k\varepsilon}{\varepsilon}\, v_i''([k\varepsilon,(k+1)\varepsilon]) &= \alpha + v([0,a]) - v_i''([0,k\varepsilon]) && \text{by (2.4)}\\
b - k\varepsilon &= \frac{\varepsilon}{v_i''([k\varepsilon,(k+1)\varepsilon])}\left(\alpha + v([0,a]) - v_i''([0,k\varepsilon])\right)\\
b &= \frac{\varepsilon}{v_i''([k\varepsilon,(k+1)\varepsilon])}\left(\alpha + v([0,a]) - v_i''([0,k\varepsilon])\right) + k\varepsilon.
\end{aligned}$$

Note that everything on the right-hand side of the equation is known at this point in the binary search, so the returned cut can be found via arithmetic methods with only three queries. The code is implemented this way to be less computationally expensive.

These two methods form the foundation of the algorithm, and all future methods heavily utilise them, so it was essential that they were written in the most computationally efficient way possible. The formulae described above accurately represent how the code was implemented in the platform.

2.3 Implementation of the Algorithm

Step 1: We first need to find the unique cut positions for each agent for which one-third of the cake lies to the right of said cut. These cut positions are guaranteed to be unique because of the hungriness modification made to the valuation functions. The agent who returns the rightmost cut is identified (this agent will hereafter be referred to as Agent 1), and we then find the division for this agent that splits the cake into thirds [2]. This is the equipartition. Because of the additive valuation assumption, this can be found by simply taking the value of the entire cake for Agent 1 and taking cut queries for one-third of this value. An example of an equipartition returned in the Fair Slice platform in the steps section of the results can be seen below.

Step 2: Once we have the equipartition, we check if this is itself an ε-envy-free division. If so, we terminate the algorithm here and return the relevant cut positions. If not, we move on to the next step. It is clear by construction of the equipartition that both Agent 2 & 3 prefer either slice 1 or slice 2. A check is run for which cases are true, and the algorithm runs the following corresponding step [2].

Step 3a: If slice 1 is preferred by Agent 2 & 3, the next step is to run a binary search over the left cut. The constraints for this binary search are as follows:

- The lower bound for the left cut position, l_{lower}, is the left bound of the cake, 0. This position for the left cut is guaranteed to return a division where Agent 2 & 3 both prefer slice 2 or 3.
- The upper bound for the left cut position, l_{upper}, is the left cut of the equipartition. This position for the left cut is guaranteed to return a division where Agent 2 & 3 both prefer slice 1.
- For each inspected left cut in the binary search, the right cut, r, is identified such that Agent 1 is indifferent between slices 1 & 2.

These constraints give two assurances. Agent 1 always weakly prefers a slice that is not slice 1, and, by the intermediate value theorem, there must be a point in this binary

search where at least one of the other agents will be indifferent between slice 1 and another slice. This guarantees that this binary search will reach a division which is ε-envy-free. The implementation of the binary search, aided by a graph utilised in the Fair Slice platform interactive course, is as follows:

1. Find the midpoint of the left cut bounds, l_{mid}, such that Agent 1's valuations $v_1''([l_{lower}, l_{mid}]) = v_1''([l_{mid}, l_{upper}])$.
2. Find the right cut, r, such that Agent 1's valuations $v_1''([l_{mid}, r]) = v_1''([r, 1])$.
3. If an ε-envy-free division exists at the division $\{l_{mid}, r\}$, terminate the algorithm and return the corresponding cuts. If not, if both Agent 2 & 3 prefer slice 1, set $l_{upper} = l_{mid}$ and continue. Otherwise set $l_{lower} = l_{mid}$ and continue [2].

The main computational problem presented in this implementation is that of a cut that returns two slices of equal value. For this, my solution was to leverage the additive valuation assumption similarly to how it was leveraged in finding the equipartition. Looking at step 1 of this implementation specifically, I took the value $x = v_1''([l_{lower}, l_{upper}])$ and then utilised a cut query, $cut(l_{lower}, x/2)$, to find l_{mid}.

Step 3b: If slice 2 is preferred by Agent 2 & 3, the next step is running a binary search over the right cut. The constraints for this binary search are as follows:

- The lower bound for the right cut position, r_{lower}, is the midpoint of the cake according to Agent 1 (i.e. the cut that satisfies $v_i''([0, r_{lower}]) = v_i''([r_{lower}, 1])$. This position for the right cut is guaranteed to return a division where Agent 2 & 3 both prefer slice 1 or 3 (given the corresponding left cut).
- The upper bound for the right cut position, r_{upper}, is the right cut of the equipartition. This position for the right cut is guaranteed to return a division where Agent 2 & 3 both prefer slice 2 (given the corresponding left cut).
- For each inspected right cut in the binary search, the left cut, l, is identified such that Agent 1 is indifferent between slices 1 & 3.

These constraints offer the same assurances as in step 3a with respect to slice 2. The implementation of this binary search, aided by a graph utilised in the Fair Slice platform interactive course, is as follows:

1. Find the midpoint of the right cut bounds, r_{mid}, such that Agent 1's valuations $v_1''([r_{lower}, r_{mid}]) = v_1''([r_{mid}, r_{upper}])$.

2. Find the left cut, l, such that Agent 1's valuations $v_1''([0, l]) = v_1''([r_{mid}, 1])$.

3. If an ε-envy-free division exists at the division $\{l, r_{mid}\}$, terminate the algorithm and return the corresponding cuts. If not, if both Agent 2 & 3 prefer slice 2, set $r_{upper} = r_{mid}$ and continue. Otherwise set $r_{lower} = r_{mid}$ and continue [2].

Computing the cut positions in this implementation can be done using similar methods as described in step 3b, though bisection is not necessary in step 2 of this implementation.

With the conclusion of these steps, an ε-envy free division will be returned. Because Branzei-Nisan wrote this algorithm with consideration for the Robertson-Webb query model, which does not assume boundedness, there is a further step after both steps 3a and 3b that will run if an envy-free division is not returned once the relevant slice is worth less than ε to Agent 1. At this point, it will start reducing this slice value relative to Agent 2's until an ε-envy free division is found [2]. This isn't necessary for our purposes, as the Fair Slice platform exclusively takes in bounded valuation functions. This ensures that reducing the value of the relevant slice relative to Agent 1's valuations must eventually return an envy-free division as a slice an ε-wide interval cannot be worth more than ε to any agent.

It is also worth noting that while the binary search as it is implemented will tend towards indifference between the relevant slice and some other slice for at least one agent, it is unnecessary to run it until it reaches this conclusion if it finds an ε-envy-free division beforehand. Any ε-envy-free division will do. This fact helps improve the runtime of this specific algorithm, but that will not be the case in future algorithms discussed.

Chapter 3

The Hollender-Rubinstein algorithm

3.1 Eval and Cut Queries

After performing the same input preprocessing as in the Brânzei-Nisan algorithm, we again must implement discretised eval and cut queries. The discretisation in this algorithm is similar to the Brânzei-Nisan algorithm but notably more complex.

- Ensure the value queries are strictly hungry (according to a different but equivalent definition $v_i([a,b]) < v_i([a',b'])\,\forall\,[a,b] \subsetneq [a',b']$ [3]) for all agents i via the following preprocessing:

  $$v_i'([a,b]) = v_i([a,b])/2 + \varepsilon \mid b-a \mid \in [0,1]$$

  This satisfies strict hungriness as a result of the monotonicity assumption (in fact, the Fair-Slice valuation functions are additive, a special case of monotone) as shown below:

  $$\begin{gathered} v_i([a,b]) < v_i([a',b']) \\ v_i([a,b])/2 + \varepsilon \mid b-a \mid < v_i([a',b'])/2 + \varepsilon \mid b'-a' \mid \\ (v_i([a,b]) - v_i([a',b']))/2 < \varepsilon(\mid b'-a' \mid - \mid b-a \mid) \end{gathered}$$

  By monotonicity of valuations ($v_i([a,b]) \leq v_i([a',b'])\,\forall\,[a,b] \subseteq [a',b']$) the LHS must be less than zero and by definition the RHS must be greater than zero, so this modified valuation function is strictly hungry.

- Then, the valuation functions are approximated via piecewise-linear functions on an ε-grid $\{0, \varepsilon, 2\varepsilon, \ldots, 1-\varepsilon, 1\}$. The ε-interval of the start cut a is identified and

upper and lower bounds are identified as $\underline{a} = \varepsilon k \leq a \leq \varepsilon(k+1) = \bar{a}$ and the same process is done to the end cut b to find $\underline{b}$ & $\bar{b}$. If $\bar{a} - a \leq b - \underline{b}$ let

$$v_i''([a,b]) = \frac{(\bar{a}-a)-(b-\underline{b})}{\varepsilon} v_i'([\underline{a},\underline{b}]) + \frac{b-\underline{b}}{\varepsilon} v_i'([\underline{a},\bar{b}]) + \frac{a-\underline{a}}{\varepsilon} v_i'([\bar{a},\underline{b}]). \tag{3.1}$$

Otherwise, let

$$v_i''([a,b]) = \frac{(b-\underline{b})-(\bar{a}-a)}{\varepsilon} v_i'([\bar{a},\bar{b}]) + \frac{\bar{a}-a}{\varepsilon} v_i'([\underline{a},\bar{b}]) + \frac{\bar{b}-b}{\varepsilon} v_i'([\bar{a},\underline{b}]). \tag{3.2}$$

These modifications are again performed as the algorithm is running, and the valuation functions retrieved from the user inputs need not be changed as preprocessing. Each value query utilised in the algorithm uses, at most, three intermediate value queries.

One interesting difference between this discretisation and the discretisation performed by Brânzei and Nisan is the lack of subtraction between value queries on intervals whose intersection is the interval of interest. This results from the Hollender-Rubinstein algorithm finding ε-envy-free divisions for monotone valuation functions, not just additive ones. This makes the algorithm more powerful but means that the implementation can no longer make use of some of the valuable shortcuts that aided the implementation of the Brânzei-Nisan algorithm [3]

Cut queries can then be implemented as a binary search of value queries like in the Brânzei-Nisan algorithm [3]. While the algorithm can utilise caching of the intermediate value queries once the ε-interval, my initial implementation did not utilise this fact, so I will discuss the specifics of this approach later in the paper.

3.2 Initial Implementation of the Algorithm

The implementation of this algorithm is conceptually similar to that of the Brânzei-Nisan algorithm but with a few key differences. As previously mentioned, the broader assumption of monotone valuation functions means that the divisions satisfying particular prerequisites will require more computation than in the Brânzei-Nisan algorithm. The Hollender-Rubinstein algorithm is also parameterised over Agent 1's value for their preferred slice, α. This allows the algorithm to test different cases in each loop where Brânzei-Nisan had to identify one of two cases (either Agents 2 & 3 preferred slice 1 or 2) before the algorithm began working within the identified case exclusively. My initial implementation is described below.

Step 1: We first need to find the unique equipartition for Agent 1. This is done by finding the unique positions of the three cuts via independent investigation. Because this initial implementation does not utilise caching of intermediate value queries, it is not strictly necessary to find the cut locations independently. Still, I implemented it this way to make the code easier to adapt later. This fact is true for all subsequent computations of multiple cut locations.

- The left cut location is found via binary search over the range $[0, 1]$. For each inspected left cut l, find the middle cut m such that $v_1''([l, m]) = v_1''([0, l])$ and subsequently the right cut r such that $v_1''([m, r]) = v_1''([0, l])$. This is achieved via successive cut queries. Then, if $v_1''([r, 1]) > v_1''([0, l])$, the updated left cut must move to the right. Otherwise, it must be moved to the left. The right cut can be found analogously.

- The middle cut is again found via binary search over the range $[0, 1]$. For each inspected middle cut m, find the left cut l such that $v_1''([0, l]) = v_1''([l, m])$ and, likewise, the right cut r such that $v_1''([m, r]) = v_1''([r, 1])$. This is achieved via what will be described as a bisection cut query, which is found via a binary search of cut locations between the start and end cut passed in (e.g. 0 & m for the location of l). Then if $v_1''([0, l]) > v_1''([r, 1])$, the updated middle cut must move to the left. Otherwise, it must move to the right.

The criterion for how to update the cut positions leverages the monotonicity and hungriness of the valuation functions. i.e. if l moves to the right, $v_i''([0, l])$ must increase and $v_i''([l, m])$ must decrease. Therefore, m must move to the right to ensure $v_i''([0, l]) = v_i''([l, m])$. This reasoning can be extended to r and utilised to inform how to update cut locations in binary searches according to given conditions.

Step 2: Once we have the equipartition, we check if this is itself an -envy-free division. If so, we terminate the algorithm here and return the relevant cut positions. If not, we move on to the main body of the algorithm.

Step 3: The main body of the algorithm is a binary search parameterised over Agent 1's value for their preferred slice, α. Hollender-Rubinstein defines an invariant with two conditions, A & B, for which at least one of the conditions must hold at the α unique to the equipartition and neither condition holds at $\alpha = 1$ by hungriness. The only exit condition of the invariant is an ε-envy-free division. This informs the structure of the body wherein if the invariant holds at the inspected α, the lower bound $\underline{\alpha}$, initially defined as the α unique to the equipartition, is updated such that $\underline{\alpha} = \alpha$. Otherwise, the

upper bound $\bar{\alpha}$, initially defined as 1, is updated such that $\bar{\alpha} = \alpha$. By construction, this binary search will converge to an α that gives an ε-envy-free division. What remains is to describe the implementation of the invariant check.

Condition A: This condition is analogous to the cases discussed in the Brânzei-Nisan algorithm. It states that, given Agent 1 values three slices at α, the remaining slice is preferred by two or more of the other agents. It is also imperative that Agent 1 does not prefer this remaining slice. In the case where slice 2 is the slice of interest, the slice locations are found via cut queries. l is found such that $v_1''([0, l]) = \alpha$, r is found such that $v_i''([r, 1]) = \alpha$ and then this value is used to find m such that $v_1''([m, r]) = \alpha$. It is then necessary to check that the returned division is valid (i.e. $l \leq r$ as there is no reason this cannot be the case with this implementation). If it is not, the algorithm moves on to one of the following cases. If the division is valid, the algorithm checks to see if at least two agents prefer the relevant slice within ε. If it is, $\underline{\alpha}$ is updated and the loop continues. If not, the algorithm moves on to another case. The other cases can be checked using similar or equivalent methods.

Condition B: This condition states that given Agent 1 values two slices at α, the remaining slices are each preferred by two or more of the other agents. It is again imperative that Agent 1 does not prefer either of the remaining slices. Given that there are only three other agents and two slices, it is also necessary that at least one agent is indifferent between the remaining slices for the condition to hold. This fact is leveraged to find the divisions for each case. There are 3 cases for condition B, which I will now describe.

1. **Relevant slices are adjacent:** For the representative case where the relevant slices are 2 and 3, l and r are found via cut queries from the outer edges 0 and 1 of value α according to Agent 1's valuation. We then iterate over each agent we will describe as indifferent and find m via a bisection cut query between l and r according to the indifferent agent's valuation. After checking that the returned division is valid, the algorithm checks that the two slices $[l, m]$ and $[m, r]$ are each preferred by at least two agents. If they are, $\underline{\alpha}$ is updated and the loop continues. If not, the algorithm proceeds to check the following cases. The other cases are handled similarly.

2. **Relevant slices are one apart:** For the representative case where the relevant slices are 1 and 3, r is found via a cut query from the rightmost edge 1 of value α according to Agent 1's valuation. We then iterate over each indifferent

agent and find l and m via binary searches. For l, the binary search has bounds $[0, r]$ and for each investigated l, we find m via a cut query with start cut l of value α according to Agent 1's valuation. The bounds for l are then updated analogously to how they were updated when finding the equipartition with the aim that $v_i''([0, l]) = v_i''([m, r])$ where i is the indifferent agent. m is found analogously. After checking that the returned division is valid, the algorithm checks that the two slices $[0, l]$ and $[m, r]$ are each preferred by at least two agents. If they are, $\underline{\alpha}$ is updated and the loop continues. If not, the algorithm proceeds to check the following cases. The other case is handled similarly.

Relevant slices are two apart: This case is handled similarly to the case where the relevant slices are one apart, so it will not be described in detail as there are no new implementation challenges. Like the previous cases, if the case holds, $\underline{\alpha}$ is updated and the loop continues. If not, given that this is the last case to check, we know the invariant does not hold for this α and $\bar{\alpha}$ is updated before continuing the loop.

One interesting feature of the algorithm is that this binary search over α continues until $|\bar{\alpha} - \underline{\alpha}| < \varepsilon^4/12$ to ensure an ε-envy-free division is returned for the original, unmodified valuation functions. An observation about this tolerance is that it necessitates an ε value that, in some cases, could be prohibitively large. In theory, the algorithm can work for whichever ε one may desire; when taking machine precision into account, the minimum ε is ~ 0.000227. This gets even larger when considering necessary preprocessing to ensure the valuation functions are 1-Lipschitz, ε is divided by whatever constant the valuation function is divided by. This is not a problem for the Fair Slice platform and is likely not an issue for most applications, but it is worth noting as a possible improvement area.

3.3 Slice Assignment

3.4 Runtime Improvements

What follows are changes to the initial algorithm implementation that Hollender-Rubinstein did not specify to improve runtime. It is essential to specify that these changes are not contrary to how Hollender-Rubinstein described the algorithm in the original paper [3] but are instead things that were not explicitly stated in the paper but

may have been implied. The runtime could've been further improved by leveraging the additive nature of the Fair Slice valuation functions, but it was important to prioritise the algorithm's integrity by not deviating from the explicit implementation choices from the paper. Specific changes to this effect will be detailed in the appendix for if they are deemed useful in the future.

- value query
- epsilon bounds (specify intermediate value theorem). Bonus in that it also returns if a cut is impossible and the loop moves on.
- Talk about the assignment in the above section.
- Talk about algorithm termination and its limitations on ε.

Chapter 4

Envy-Free Moving Knife Algorithm for Five Agents

Condition A: Agent 1 is indifferent between its four favourite slices, and the remaining piece is (weakly) preferred by at least two of the four remaining agents.

Condition B: Agent 1 is indifferent between its three favourite slices, and the remaining two slices are each (weakly) preferred by at least two of the four remaining agents, with one of these agents being indifferent between the two.

Condition C: Agent 1 is indifferent between its three favourite slices, and the remaining two slices are each (weakly) preferred by at least two of the four remaining agents, with one of these agents being indifferent between one of Agent 1's favourite slices and one of the remaining two slices.

Condition D: Agent 1 is indifferent between its two favourite slices, and the remaining three slices are each (weakly) preferred by at least two of the four remaining agents, with two of these agents being indifferent between one of the remaining slices not preferred by the other and a mutual remaining slice.

Condition E: Agent 1 is indifferent between its two favourite slices, and the remaining three slices are each (weakly) preferred by at least two of the four remaining agents, with one of these agents being indifferent between the three.

4.1 The algorithm

Begin by having Agent 1 split the cake into fifths according to their valuations. If this division is envy-free, return it. If not, begin at condition A (Note that if the division is not envy-free, condition A must be true) and increase the size of Agent 1's four

favourite slices such that Agent 1 remains indifferent between them. Continue with this until condition A can no longer be true if the process is continued. At this point, continuing the trail from another condition must be possible, or an envy-free division must exist. For every condition, if continuing the trail for this condition is no longer possible, another trail can be pivoted to, or an envy-free division must exist. I will prove this in the following section.

4.2 Proof of Correctness

First, we will state the formal definitions of the conditions.

Condition A: There exists a slice $k \in \{1, 2, 3, 4, 5\}$ such (i) that for all slices $t, t' \in \{1, 2, 3, 4, 5\} \setminus \{k\}$ such that $v_1(t) = v_1(t') \geq v_1(k)$, and (ii) there exists two agents $i, i' \in \{2,3,4,5\}$ such that $v_i(k) \geq \max_t v_i(t)$ and $v_{i'}(k) \geq \max_t v_{i'}(t)$.

Condition B: There exists two slices $k, k' \in \{1, 2, 3, 4, 5\}$ such (i) that for all slices $t, t' \in \{1, 2, 3, 4, 5\} \setminus \{k, k'\}$ such that $v_1(t) = v_1(t') \geq \max\{v_1(k), v_1(k')\}$, and (ii) for each slice $h \in \{k, k'\}$ there exists two agents $i, i' \in \{2,3,4,5\}$ such that $v_i(h) \geq \max_t v_i(t)$ and $v_{i'}(h) \geq \max_t v_{i'}(t)$ with (iii) one of these agents satisfying $v_i(k) = v_i(k') \geq \max_t v_i(t)$.

Condition C: There exists two slices $k, k' \in \{1, 2, 3, 4, 5\}$ such (i) that for all slices $t, t' \in \{1, 2, 3, 4, 5\} \setminus \{k, k'\}$ such that $v_1(t) = v_1(t') \geq \max\{v_1(k), v_1(k')\}$, and (ii) for each slice $h \in \{k, k'\}$ there exists two agents $i, i' \in \{2,3,4,5\}$ such that $v_i(h) \geq \max_t v_i(t)$ and $v_{i'}(h) \geq \max_t v_{i'}(t)$ with (iii) one of these agents satisfying $v_i(k) = v_i(t)$ for some slice $t \in \{1, 2, 3, 4, 5\} \setminus \{k, k'\}$.

Condition D: There exists three slices $k, k', k'' \in \{1, 2, 3, 4, 5\}$ such (i) that for all slices $t, t' \in \{1, 2, 3, 4, 5\} \setminus \{k, k', k''\}$ such that $v_1(t) = v_1(t') \geq \max\{v_1(k), v_1(k'), v_1(k'')\}$, and (ii) for each slice $h \in \{k, k', k''\}$ there exists two agents $i, i' \in \{2,3,4,5\}$ such that $v_i(h) \geq \max_t v_i(t)$ and $v_{i'}(h) \geq \max_t v_{i'}(t)$ with (iii) two of these agents satisfying $v_i(k) = v_i(k')$ and $v_{i'}(k'') = v_{i'}(k')$.

Condition E: There exists three slices $k, k', k'' \in \{1, 2, 3, 4, 5\}$ such (i) that for all slices $t, t' \in \{1, 2, 3, 4, 5\} \setminus \{k, k', k''\}$ such that $v_1(t) = v_1(t') \geq \max\{v_1(k), v_1(k'), v_1(k'')\}$, and (ii) for each slice $h \in \{k, k', k''\}$ there exists two agents $i, i' \in \{2,3,4,5\}$ such that $v_i(h) \geq \max_t v_i(t)$ and $v_{i'}(h) \geq \max_t v_{i'}(t)$ with (iii) one of these agents satisfying $v_i(k) = v_i(k') = v_i(k'')$ and $v_{i'}(k'') = v_{i'}(k')$.

Condition A proof: If we reach the point of increasing the value of Agent 1's favourite slices (we will call this a Trail A), we must have reached a point where

condition A will no longer hold, it must be because one of the two remaining agents 2 & 3, say Agent 3 wlog, will no longer prefer the remaining slice, k, and will instead prefer another slice k' if we continue along this trail (that is, Agent 3 is currently indifferent between k and k'. There are a few cases of this:

- Agents 4 & 5 prefer different slices $t, t' \notin \{k\}$ and Agent 3 prefers some slice $k' \notin \{t, t', k\}$, meaning this division is envy-free and the algorithm terminates
- Agents 4 & 5 prefer different slices $t, t' \notin \{k\}$ and Agent 3 prefers some slice $k' \in \{t, t'\}$, meaning this division satisfies condition B with agent 3 being the indifferent agent and we can continue along Trail B.
- Agents 4 & 5 prefer the same slice $t \notin \{k\}$ and Agent 3 prefers some slice $k' = t$, meaning this division again satisfies condition B with agent 3 being the indifferent agent and we can continue along Trail B.
- Agents 4 & 5 prefer the same slice $t \notin \{k\}$ and Agent 3 prefers some slice $k' \neq t$, meaning this division satisfies condition C with agent 3 being the indifferent agent and we can continue along Trail C.

Condition B proof: If we reach the point on Trail B, we must have reached a point where condition B will no longer hold. This is because either (i) Agent 1 is currently indifferent between its three favourite slices and one of the remaining slices, (ii) the indifferent Agent, say Agent 5 wlog, is now indifferent between the two remaining slice k and k' as well as a third slice $k'' \notin \{k, k'\}$, or (iii) one of the remaining agents is currently indifferent between one of the remaining slices k and some other slice $t \neq k$.

In case (i), Agent 1 is indifferent between its four favourite slices, and at least two agents prefer the remaining slice. This aligns with condition A, and we can continue along trail A.

In case (ii), there are the following scenarios to consider:

- Agents 2 & 3 prefer the same slice k and Agent 4 prefers slice k'. In this case, Agent 1 is indifferent between their three favourite slices and the two remaining slices are each preferred by two agents, one of which is indifferent between one of these slices and one of Agent 1's favourite slices. This is in line with condition C, and we can continue along Trail C with agent 5 being held indifferent. An important note here is that, when continuing along trail C, Agent 5 can only be indifferent between some slice $k'' \notin \{k, k'\}$ and one of the slices k, k' as the other slice must monotonically decrease in value as the other four increase in value.

- Agents 2 prefers slice k, Agent 3 prefers slice k', and Agent 4 prefers some slice $t \notin \{k, k', k''\}$. This is an envy-free division, and the algorithm terminates.

- Agents 2 prefers slice k, Agent 3 prefers slice k', and Agent 4 prefers some slice k''. In this case, Agent 1 is indifferent between its two favourite slices and the remaining three slices are each preferred by at least two agents, one of which is indifferent between the three slices. This satisfies Condition E, and we can continue along Trail E with Agent 5 held indifferent.

In case (iii), there are the following scenarios to consider:

- Agents 2 & 3 prefer the same slice k' and Agent 4 is indifferent between slices k and k'. In this case, we can pivot to a different trail B with Agent 4 held indifferent. (Must prove exhaustively for all cases, but should be identical to the same proof in Hollender-Rubinstein). This is analogous to the case where Agent 2 prefers slice k, Agent 3 prefers slice k', and Agent 4 is indifferent between slices k and k' and the case where Agent 3 prefers slice k', Agent 4 is indifferent between slices k and k' with Agent 1 preferring some slice $t \notin \{k, k'\}$

- Agents 2 & 3 prefer the same slice k' and Agent 4 is indifferent between slices k and some slice $t \notin \{k, k'\}$. In this case, Agent 1 is indifferent between their three favourite slices and the two remaining slices are each preferred by two agents, one of which is indifferent between one of these slices and one of Agent 1's favourite slices. This satisfies Condition E, and we can continue along Trail C with Agent 4 held indifferent

- Agent 3 prefers slice k' and Agent 4 is indifferent between slices k and some slice $t \notin \{k, k'\}$, with Agent 2 also preferring slice t. In this case, Agent 1 is indifferent between its two favourite slices, with the three remaining slices each preferred by at least two of the four remaining agents, with two of these agents being indifferent between one of the remaining slices not preferred by the other and a mutual remaining slice. This is in line with condition D, and we can continue along Trail D with Agents 4 & 5 being held indifferent.

- Agent 3 prefers slice k' and Agent 4 is indifferent between slices k and some slice $t \notin \{k, k'\}$, with Agent 2 preferring some slice $t' \notin \{k, k', t\}$. This is an envy-free division, and the algorithm terminates.

Condition C proof: If we reach the point on Trail C, we must have reached a point where condition C will no longer hold. This is because either (i) Agent 1 is currently indifferent between its three favourite slices and one of the remaining slices, (ii) the agent that prefers the remaining slice k along with the indifferent agent is now indifferent between k and some slice $t \notin \{k, k'\}$ (note that they cannot prefer slice k' as it is monotonically decreasing in value), or (iii) one of the agents that prefers the remaining slice k' not preferred by the indifferent agent is now indifferent between k' and some slice $t \neq k'$.

In case (i), Agent 1 is indifferent between its four favourite slices and the remaining piece is preferred by at least two of the four remaining agents. This is in line with condition A, and we can continue along Trail A. Note that Agent 1 cannot be indifferent between its three favourite slices, and the remaining slice not preferred by the indifferent agent k' because its value is monotonically decreasing.

In case (ii), the agent that prefers the remaining slice k along with the indifferent agent and is now indifferent between k and some slice $t \notin \{k, k'\}$, say Agent 4, is now a valid choice for the indifferent agent if we were to change to a different trail C. This is because, continuing along this new trail C with Agent 4 held indifferent, Agent 5 will no longer weakly prefer one of Agent 1's favourite slices. The fact that continuing along Trail C with Agent 5 held indifferent would result in condition C no longer being true means that, instead, holding Agent 4 indifferent would result in the size (and therefore value, by monotonicity and hungriness) of the slice k increasing quicker than if Agent 5 was held indifferent. This means that Agent 5 will strictly prefer slice k in this new trail C, and condition C will continue to be satisfied. (Need to prove rigorously)

In case (iii), there are the following scenarios to consider:

- Given Agent 3 prefers slice k', Agent 4 prefers slice k, Agent 5 is indifferent between k and some slice $t \notin \{k, k'\}$, and Agent 2 is indifferent between slices k' and k, this now satisfies condition B, so we can continue on trail B with agent 2 held indifferent.

- Given Agent 3 prefers slice k', Agent 4 prefers slice k, Agent 5 is indifferent between k and some slice $t \notin \{k, k'\}$, and Agent 2 is indifferent between slices k' and t, Agent 1 is indifferent between its two favourite slices, and the remaining three slices are each (weakly) preferred by at least two of the four remaining agents, with two of these agents being indifferent between one of the remaining slices not preferred by the other and a mutual remaining slice. This satisfies

condition D, so we can continue along trail D with Agents 2 & 5 held indifferent.

- Given Agent 3 prefers slice k', Agent 4 prefers slice k, Agent 5 is indifferent between k and some slice $t \notin \{k, k'\}$, and Agent 2 is indifferent between slices k' and some slice $t' \notin \{k, k', t\}$, this is an envy-free division and the algorithm terminates.

Condition D proof: If we reach the point on Trail D, we must have reached a point where condition D will no longer hold. This is because either (i) Agent 1 is currently indifferent between its two favourite slices and one of the remaining slices, (ii) one of the agents that is indifferent between slices k and k' also weakly prefers some slice $t \notin \{k, k'\}$, or (iii) one of the remaining agents is now indifferent between their preferred slice k and some slice $t \neq k$.

In case (i), there are the following scenarios to consider:

- Agent 1 is indifferent between its two preferred slices and the slice that the indifferent agents mutually prefer, say Agents 4 & 5 wlog, slice k'. In this case, Agent 1 is indifferent between its three favourite slices, and the remaining two slices are each (weakly) preferred by at least two of the four remaining agents, with one of these agents being indifferent between one of Agent 1's favourite slices and one of the remaining two slices. This satisfies condition C and we can continue along Trail C with either Agent 4 or 5 held indifferent, depending on which indifferent agent would result in the other strictly preferring slice k or k'' and not k'.

- Agent 1 is indifferent between its two preferred slices and one of the other slices, say slice k, preferred by Agents 3 & 4, with Agent 4 also weakly preferring slice k'. In this case, Agent 1 is indifferent between its three favourite slices, and the remaining two slices are each (weakly) preferred by at least two of the four remaining agents, with one being indifferent between them. This satisfies condition B, and we can continue along trail B with Agent 4 being held indifferent.

In case (ii), there are the following scenarios to consider:

- Given Agent 3 prefers slice k, Agent 2 prefers slice k', Agent 4 is indifferent between slices k'' and k' and Agent 5 is indifferent between slices k, k', and k'', this now satisfies condition E and we can continue along Trail E with Agent 5 held indifferent. We know Agent 4 will strictly prefer slice k'' on this trail after switching (Need to explain rigorously.)

- Given Agent 3 prefers slice k, Agent 2 prefers slice k', Agent 4 is indifferent between slices k'' and k' and Agent 5 is indifferent between slices k, k', and some slice $t \notin \{k, k'\}$, this is an envy-free division and the algorithm terminates.

In case (ii), there are the following scenarios to consider:

- Given Agent 3 prefers slice k, Agent 4 is indifferent between slices k'' and k' and Agent 5 is indifferent between slices k, k', and Agent 2 is indifferent between slice k'' and slice k, this still satisfies condition D, and we can continue on a new trail D with Agents 2 and 4 held indifferent. Agent 5 will strictly prefer slice k' when the trail continues (must explain rigorously.)

- Given Agent 3 prefers slice k, Agent 4 is indifferent between slices k'' and k' and Agent 5 is indifferent between slices k, k', and Agent 2 is indifferent between slice k' and slice k, this still satisfies condition D, and we can continue on a new trail D with Agents 2 and 5 held indifferent. Agent 4 will strictly prefer slice k'' when the trail continues (must explain rigorously.)

- Given Agent 3 prefers slice k, Agent 4 is indifferent between slices k'' and k' and Agent 5 is indifferent between slices k, k', and Agent 2 is indifferent between slice k' and slice $t \notin \{k, k', k''\}$, this is an envy-free division, and the algorithm terminates.

Condition E proof: If we reach the point on Trail E, we must have reached a point where condition E will no longer hold. This is because either (i) Agent 1 is currently indifferent between its two favourite slices and one of the remaining slices, (ii) The agent that is indifferent between slices k, k', and k'' also weakly prefers some slice $t \notin \{k, k', k''\}$, or (iii) one of the remaining agents is now indifferent between their preferred slice k and some slice $t \neq k$.

For case (i), this division now satisfies condition B, and we can continue along trail B with the same agent held indifferent.

For case (ii), this is an envy-free division, and the algorithm terminates.

For case (iii), there are the following scenarios to consider:

- Given Agent 5 is indifferent between slices k, k', and k'', Agent 4 prefers slice k'', Agent 3 prefers slice k', and Agent 2 is indifferent between slice k and k', this now satisfies condition D, and we can continue along trail D with Agents 2 and 5 held indifferent. On this trail, Agent 5 will no longer weakly prefer slice k'. Must prove rigorously.

- Given Agent 5 is indifferent between slices k, k', and k'', Agent 4 prefers slice k'', Agent 3 prefers slice k', and Agent 2 is indifferent between slice k and some slice $t \notin \{k, k', k''\}$, this is an envy-free division, and the algorithm terminates.

Chapter 5

Conclusions

5.1 Final Reminder

The body of your dissertation, before the references and any appendices, *must* finish by page 40. After the preliminary material, the introduction should have started on page 1.

You may not change the dissertation format (e.g., reduce the font size, change the margins, or reduce the line spacing from the default 1.5 spacing). Be careful if you copy-paste packages into your document preamble from elsewhere. Some LaTeX packages, such as `fullpage` or `savetrees`, change the margins of your document. Do not include them!

Over-length or incorrectly-formatted dissertations will not be accepted, and you would have to modify your dissertation and resubmit. You cannot assume we will check your submission before the final deadline, and if it requires resubmission after the deadline to conform to the page and style requirements, you will be subject to the usual late penalties based on your final submission time.

Bibliography

[1] Simina Brânzei and Noam Nisan. Communication complexity of cake cutting. *CoRR*, abs/1709.09876, 2017.

[2] Simina Brânzei and Noam Nisan. The query complexity of cake cutting. *ArXiv*, abs/1705.02946, 2017.

[3] Alexandros Hollender and Aviad Rubinstein. Envy-free cake-cutting for four agents, 2023.

Appendix A

First appendix

A.1 First section

Any appendices, including any required ethics information, should be included after the references.

Markers do not have to consider appendices. Make sure that your contributions are made clear in the main body of the dissertation (within the page limit).

Appendix B

Participants' information sheet

If you had human participants, include essential information that they were given in an appendix, and point to it from the ethics declaration.

Appendix C

Participants' consent form

If you had human participants, include information about how consent was gathered in an appendix, and point to it from the ethics declaration.