On the Complexity of the Chamberlin–Courant Rule Under Generalized Single-Crossing Preferences

Andrei Constantinescu
Supervisor: Edith Elkind

3rd Year Project Report
Honour School of Computer Science (Part B)

Trinity Term 2020

Abstract

We drive new results for the problem of determining winning committees for the Chamberlin–Courant (CC) rule in the case of generalized single-crossing electorates. For the classical case, we improve the $O(n^2mk)$ algorithm of Skowron et al. [30] to $O(nmk)$. Then, using novel techniques, we further improve the time bounds to $nm2^{O\left(\sqrt{\log k \log\log n}\right)}$ for the utilitarian case with $k = \Omega(\log n)$ and $O(nm \log n \log(nm))$ for the egalitarian case. For tree single-crossing elections, we point out an error in an earlier attempt by Clearwater, Puppe and Slinko [8] and propose the first polynomial time algorithm for the problem, running in time $O(nmk)$. Finally, we begin the study of CC on profiles which are single-crossing on median graphs. For this purpose, we consider CC on 2-dimensional grids (although our ideas easily generalize to higher dimensions), for which we formulate a conjecture under which a polynomial algorithm becomes feasible. We conclude by showing that, even under no additional assumptions, this polynomial-time algorithm is actually at least a good approximation, in a sense which we define.

Acknowledgements

I want to express my endless gratitude to Edith Elkind for being my supervisor and guiding me through this highly rewarding and intriguing experience of understanding the field of Computational Social Choice. She has provided me with plenty of advice on how to best focus my efforts and has taken countless hours to carefully review this manuscript.

I also extend the warmest commendation to my tutors: Tom Melham, Coralia Cartis and Ohad Kammar, for guiding me through the years of undergraduate study and teaching me many important lessons, both academically and beyond.

I want to thank Arkadii Slinko for the the useful communication about the algorithm for the Chamberlin–Courant rule on tree single-crossing preferences. Likewise, I need to thank Bogdan and Nathan for proofreading parts of this work and commenting on it.

Last but not least, I want to thank my family for their never-ending love and support, being understanding throughout all phases of my academic study and always encouraging me to strive for the absolute.

Contents

1. Introduction

For most people, voting is about electing a candidate to be the leader of a society (e.g. the president of a country or of a governing body). However, the extent to which it is present in our lives is much larger—it occurs whenever a team needs to pick a good time for a meeting or a group of referees have to pick a winner for a contest (Faliszewski et al. [15]). One can go even further and think about a single Oxford student having to pick a room in their college from multiple alternatives. Of course, there will be multiple factors to weigh into the choice: price, inhabitable area, quality of heating, etc. In this case, one can think of the different factors as voters and of the rooms as candidates.

The inherent difficulty of voting comes from the fact that different elections require totally different voting rules. For example, in voting for a meeting time, *Approval* is commonly used, in which the winning time is the one when most voters could attend, while in voting for a president a multitude of more complex rules are used in practice. For example, in *Plurality*, the winning candidate is the one which is the first choice of the greatest number of voters, while the *Instant-runoff* rule repeatedly eliminates the candidate ranked first by the lowest number of voters until one candidate gains majority. Notice that the latter requires the preferences of the voters to be cast through *ordinal ballots* (fully ordered lists of candidates), unlike the in former, where only the top-rated candidate is required from each voter. Plurality is generally regarded inferior to other rules (Laslier [19]), one of the main reasons being that whenever multiple candidates are similar in the eyes of the electorate, votes for them will be roughly evenly split, reducing their chances of winning—an illustrative example comes from Pilkington [25]:

Activity	**Number of First-Choices**
Camping and Hiking in Colorado	2
Camping and Hiking in California	2
Camping and Hiking in Washington	2
Camping and Hiking in Ireland	1
Disney World	3

Table 1: Camping and Hiking should win 7-3 to Disney World, but the latter wins by Plurality.

However, even for rules with no such immediate flaws, different rules may produce totally different winners, rendering each rule more appropriate for a different scenario. Faliszewski et al. [15] state that when choosing/designing a rule there is a tension between the desire to select a candidate judged as highly as possible; i.e. not within the last few choices of too many voters, and one supported by as many voters as possible; i.e. within the first choices of many voters. They argue that presidential elections lean more towards the latter objective, since it is acceptable to have large dissatisfied minorities, while picking a convenient meeting time is more concerned

with the former, since it is perfectly fine to pick a time that is not optimal for anyone, as long as everyone can attend.

1.1 Multi-Winner Rules

The election rules discussed above are also known as *single-winner rules*. Next, we talk about *multi-winner rules*, where the goal is to select a committee of k candidates based on the voters' preferences. The practical usage of multi-winner rules is even more diverse, applying to parliamentary elections, shortlisting (e.g. candidates to interview for a job) and various business decisions (e.g. an internet store deciding which products to show on their homepage). This time, rules can be even more finely classified based on their purpose, Elkind et al. [14], Faliszewski et al. [15] discussing three main types:

- **Excellence-Based Elections**. Mostly correspond to shortlisting tasks, where a number of experts have to pick a group of k finalists. They are generally characterized axiomatically by having a property called *committee monotonicity*, meaning that if a candidate is included in the winning k-committee, then it is also included in the winning $(k+1)$-committee. This property essentially reduces these elections to the single-winner case, since the job of selecting k candidates can be done incrementally, always picking the best candidate from the remaining ones.
- **Selecting a Diverse Committee**. Unlike with Excellence-Based Elections, committees are now more than just collections of their members and there is generally little utility in electing two very similar candidates to the committee. For example, consider a variant of the Facility Location Problem[1]: in a city there are a number of households and a number of potential locations for new fire stations, the households ranking the fire stations based on how close they are to them. While locations in the city centre make for near-optimal 1-committees, it would be much wiser to spread out the fire stations to get good larger committees. Generally speaking, work on characterizing axiomatically what is meant by "diverse" is less developed, the closest notion to what interests us being that of *representation-focused* rules (Elkind et al. [14]).
- **Proportional Representation**. Informally, the goal is to reflect "all shades of political opinion" (Black [6]). This is achieved by assigning each voter to a candidate in the committee that shares their views. Parliamentary elections are a great example of this kind. Some authors require such rules to obey the additional property that each candidate represents roughly the same number of voters. However, as we will see in a moment, this is not always the case.

Multi-winner rules can also be classified based on the underlying mathematical model and/or principles. In this sense, Faliszewski et al. [15] discuss three types of rules: committee scoring rules, approval-based rules and rules based on the Condorcet principle. In the first and third, the ballots are cast as ordered lists of preferences (so-called *ordinal ballots*), while in the second one voters each give a list of "acceptable" candidates (so-called *approval ballots*). In this work we focus exclusively on the first of the three. Importantly, the way we adopt this notion generalizes what was described by Faliszewski et al. Subsequently, let a *committee scoring function* be a function

[1] See Zanjirani and Hekmatfar [31] for a detailed discussion of facility location problems.

which assigns a real number to each possible committee of size k, then by a *committee scoring rule* we understand one whose winning committees minimize/maximize a certain committee scoring function. A number of examples, in increasing order of complexity, illustrate this definition:

- **Single Non-Transferable Vote (SNTV)**. A committee's score is equal to the total number of voters whose top rated candidate is in the committee.
- **Bloc**. Each voter adds to the score of the committee a quantity equal to how many of their top k preferences are in the committee.
- **Borda**. Each voter adds to the score of the committee the positions in their list of preferences of all candidates in the committee.
- **Chamberlin–Courant (β-CC)**. Each voter adds to the score of the committee the position in their list of preferences of their highest ranked member of the committee (introduced by Chamberlin and Courant [7] in 1983).
- **Monroe**. Similar to the Chamberlin–Courant rule, but additionally requires that the number of voters each candidate represents differs by at most 1 (introduced by Monroe [23] in 1995).

One major practical difference between single-winner and multi-winner rules is in their computational complexity: while single-winner rules are, with some exceptions (see Faliszewski et al. [15] for a list of examples), computable in polynomial time[2], there is a fair number of multi-winner rules for which winner determination turns out to be NP-hard. While it is almost trivial to come up with polynomial-time algorithms for SNTV, Bloc and Borda, Procaccia et al. [26] show that computing winners for β-CC and Monroe's rule is NP-hard. There are several ways to go about mitigating this hardness. One is to use approximation algorithms, path successfully explored by Skowron et al. [29] and Lu and Boutilier [20]. The former remark that, while approximating the result of voting works great for optimization problems (such as the Facility Location problem), where nearly optimal committees are more than enough, one should be careful when doing so in a political scenario, since it is conceivable that a different committee is found to be better after the winners have been announced. Another solution considers fixed parameter tractability (FPT), where the parameter is the number of voters, candidates, or members of the committee. Betzler, Slinko and Uhlmann [4] explore this idea, producing such FPT algorithms which are fast when either the number of voters or candidates is small. Yet another solution is restricting the domain of preferences: i.e. assuming that not every combination of preferences of the voters is possible. Such preferences are called *structured* and are the subject of the next section.

1.2 Structured Preferences

In most practical situations, the preferences of the voters will be quite far from random, there being some form of structure to them. For this reason, understanding the properties of various voting rules under different ways of modelling structuredness plays a big role in modern social choice. The literature most commonly considers two structured preference domains: *single-peaked* and *single-crossing* preferences. The former were introduced by Black [5] and further popularized by Arrow [2][3], while the latter first appeared in the work of Roberts [27] on income tax schedules.

[2]This is trivially the case for the Plurality and Instant-runoff rules discussed previously.

[3]We remark that this seminal work of Kenneth Arrow, dating from 1951, culminates with what is today known as Arrow's Impossibility Theorem, which started the field of modern Social Choice Theory.

The original reason for their introduction was resolving the famous *Condorcet Paradox*, regarding the Majority voting rule: it may be the case that opinions in a society are split so that a majority of people prefer candidate A to candidate B, a majority of people prefer candidate B to candidate C, but, intransitively, a majority of people prefer candidate C to candidate A, as in the following example:

v_1	v_2	v_3
a	b	c
b	c	a
c	a	b

Figure 1: Example of preference lists of voters v_1, v_2, v_3 over candidates a, b, c, most preferred candidates being at the top. Observe that v_1 and v_3 prefer a to b, v_1 and v_2 prefer b to c and v_2 and v_3 prefer c to a, leading to a cycle $a \succ b \succ c \succ a$ in the majority.

Both single-peaked and single-crossing elections can be shown not to lead to such majority cycles and possess many other attractive properties (see Elkind, Lackner and Peters [13] for a survey of structured preferences). From the computational point of view, the assumptions of single-peakedness or single-crossingness of the preferences makes many problems, which are hard in the general case, tractable: notably, winner determination for the Chamberlin–Courant rule becomes easy while Monroe's rule remains hard for both domain restrictions (Betzler, Slinko and Uhlmann [4], Skowron et al. [30]).

Following Fischer et al. [16], an election over a finite set of candidates is called *single-crossing* if we can totally order the voters into a list such that, as we sweep through it from left to right, the relative order of every pair of candidates changes at most once (see Figure 2 for a geometric intuition). This condition may appear quite arbitrary at first, so it helps to see how it can naturally arise in practice: consider a society of office workers voting for the best temperature to have in the workspace from a proposed set of choices (e.g. 17°C, 20.4°C, 23°C). It is then natural to assume that each voter has an "optimal" temperature t_{opt} that they would be most comfortable at and that they dislike other temperatures t based on how far t is from t_{opt} (i.e. the Euclidean distance $|t - t_{opt}|$). It is now easy to see that ordering voters by t_{opt} satisfies the single-crossing condition: if t_1 and t_2 are candidate temperatures, then voters will prefer t_1 to t_2 or vice-versa depending on how their t_{opt} compares to the midpoint of $[t_1, t_2]$. This example occurs in many other forms in the literature: consider voters voting for income tax (Mirrlees [22]), or even voters in an election deciding their preferences based on a political spectrum (e.g. the left-right axis).

v_1	v_2	v_3	v_4	v_5
a	b	b	d	d
b	a	d	b	c
c	d	a	c	b
d	c	c	a	a

Figure 2: Example of single-crossing preferences of voters v_1, v_2, v_3, v_4, v_5 over candidates a, b, c, d. When the voters are ordered naturally, the "trajectories" of any two candidates cross at most once. Picture from [17].

It is important to understand that, while single-crossing elections capture this idea of one-dimensionality through orderings of the voters, it would be mistaken to believe that all single-crossing elections are of the from captured in the temperatures example (i.e. that there is a way to assign real numbers to voters and candidates such that voters prefer candidates based on the Euclidean distance)—elections like this form a strict subset of single-crossing elections, but are, nevertheless, useful for understanding the general case. Importantly for our work, single-crossing preferences can be generalized in the following manner: instead of ordering the voters into a list, we can place voters in the vertices of a graph and require that, as we walk along certain simple paths, the preferences we get are single-crossing in the classical sense. This idea was first proposed by Demange in [12]. We leave the exact details for later, but note that, depending on the shape of the graphs used, we give names to the emerging domains of preferences accordingly: preferences single-crossing on a path[4], a tree, a cycle, a median graph, etc.

Since they will not make our object of study, we will not go into the specific definition of single-peaked preferences and just note that they are yet another way to say that preferences are, in a different sense, one-dimensional.

The main objection which is often brought up when discussing structured preferences defined in such simple terms is that these definitions are not resilient to small anomalies: e.g. it often takes a single voter with unusual preferences to make an election unstructured. Indeed, as Jaeckle, Peters and Elkind point out in [17], no real-world preferences present in the PrefLib[5] library are single-crossing or single-peaked. For this reason, a lot of work has been done on recognizing preferences which are "close" to being structured[6] (see Jaeckle, Peters and Elkind [17] for a survey of such results) so as to allow extending results about such ideally structured preferences to real elections, which have less structure.

1.3 Our Contribution

The Chamberlin–Courant and Monroe rules previously discussed are some of the most studied ways of achieving Proportional Representation (e.g. Skowron et al. [30]). We note that the original definitions given above have since been extended to distinguish between *utilitarian* and *egalitarian* versions of each rule (we will make this precise in the next section). In this work we focus on the hardness of winner determination for the Chamberlin–Courant rule, in the case of generalized single-crossing elections. In what follows, let n denotes the number of voters, m the number of candidates and k the size of the sought committee. We contribute with three kinds of new results on computing winning committees for the Chamberlin–Courant rule:

- For classical single-crossing electorates, we improve on the $O(n^2mk)$ dynamic programming algorithm of Skowron et al. [30] by exhibiting an improved algorithm which runs in time $O(nmk)$. Afterwards, we introduce the Concave Monge DAG k-Link Path Problem and show how the utilitarian version of the rule reduces to it. This allows us to apply powerful algorithms which have been developed for this problem (Bein, Larmore and Park [3], Aggrawal, Schieber and Tokuyama [1], Schieber [28]) to get better time bounds of $nm2^{O\left(\sqrt{\log k \log\log n}\right)}$ for the utilitarian case and $O(nm \log n \log(nm))$ for the egalitarian one. One peculiar aspect

[4]The same as classical single-crossing preferences.

[5]See Mattei and Walsh [21] for an introduction to PrefLib (`http://www.preflib.org/`).

[6]One possible notion of closeness is that it is possible to remove a small number of voters/candidates such that the resulting election becomes structured in the usual sense. However, many others have been considered.

of our reduction is that its correctness is proven by reducing to a finite number of cases. Each of these cases can be captured by a small linear program, and the infeasibility of this program is then checked using the Z3 Theorem Prover.[7]

- For tree single-crossing electorates, Clearwater, Puppe and Slinko [8] recently proposed an algorithm for winner determination which they claimed runs in polynomial time. Unfortunately, this turns out not to be the case. We show why this is not the case and give as resolution a more complex algorithm which runs in time $O(nmk^2)$. Through careful analysis, we show that our algorithm actually runs in time $O(nmk)$.
- We consider for the first time preferences which are single-crossing on median graphs. In particular, we focus on preferences which are single-crossing on grid graphs (i.e. graphs whose vertices are points in a d-dimensional rectangular lattice). For this case, we give an algorithm which we conjecture determines a Chamberlin–Courant winning committee in polynomial time and prove that it at least serves as a good approximation.[8]

[7] `https://github.com/Z3Prover/z3`

[8] Technically, we show that our algorithm is a $(k^{d-1}, 1)$ bi-criteria approximation algorithm—we choose not to use this specialized terminology.

2. Preliminaries

Consider a society of n *voters* V with preferences over a set of m *candidates* C. The sets V and C need not be disjoint, but for all practical purposes they can be regarded as such. In this work, we will exclusively consider preferences expressed through so-called *ordinal ballots*:

Definition 2.1. The *preference order* of voter v is a linear order $\succ_v$, where $c_0 \succ_v c_1$ indicates that voter v prefers candidate c_0 to candidate c_1. This order can be regarded as the ballot cast by voter v. An n-tuple of such preferences, one for each voter, is called a *preference profile* and represents the preferences of the electorate as a whole.

Definition 2.2. For any non-negative integer n, we let $[n]$ denote the set $\{1, 2, \ldots, n\}$. Similarly, we let $[i : j]$ denote the set $\{i, i+1, \ldots, j\}$.

Definition 2.3. Let $pos_v(c)$ denote the position of candidate c in the ranking of voter v; i.e. the top-ranked candidate gets position 1, the second best position 2, etc. Additionally, denote $pos_v^{-1} : [m] \to C$ by σ_v.

Intuitively, $\sigma_v(i)$ denotes the candidate at position i in v's ranking, so that the following holds:

$$\sigma_v(1) \succ_v \sigma_v(2) \succ_v \ldots \succ_v \sigma_v(m) \tag{2.0.1}$$

All voting rules that we will consider are specialized for proportional representation. As a result, we adopt the following specific definition of multi-winner rules:

Definition 2.4. Given an electorate with preferences over a set of candidates and a fixed positive integer $k \leq m$, a *multi-winner rule* outputs a k-member committee, which is a subset of C, and assigns to each voter a candidate from the committee. A candidate which gets assigned to a voter is called that voter's *representative*.

Intuitively, representatives are to represent the interests of the voters they are assigned to. Additionally, the rules we will consider are *committee scoring rules*, in that for each potential committee and assignment of representatives, a score is computed as follows: each voter expresses how dissatisfied they are with their assigned representative through a non-negative real number, then these individual dissatisfactions are aggregated, and the resulting value becomes the score of the committee. Winning committees are the ones with the lowest score. We now formalize this intuition and introduce two versions of the Chamberlin–Courant winning rule, which we are ultimately interested in.

Definition 2.5. Given a preference profile, a mapping $r \colon V \times C \to \mathbb{Q}_+$[1] is called a *misrepresentation function* if for any voter v and candidates c_0, c_1 the condition $pos_v(c_0) < pos_v(c_1)$ implies

[1] Note that this does includes 0. We remark that $\mathbb{R}_+$ could have been used instead, but that would exceed the realm of functions which can be represented by a computer.

$r(v, c_0) \leq r(v, c_1)$. The value $r(v, c)$ is called the *dissatisfaction* of voter v with candidate c.

For our purposes, we assume that all misrepresentation functions are computable in constant time.

Definition 2.6. Let $w\colon V \to C$ denote a function that assigns voters to their representatives; i.e. under this assignment voter v is represented by candidate $w(v)$. If the image of V under w has size $|w(V)| \leq k$, then we call w a k-assignment. Additionally, define the total dissatisfaction $\Phi(V, w)$ of the given election under w by:

1. $\Phi(V, w) = \sum_{v \in V} r(v, w(v))$ in the utilitarian case.
2. $\Phi(V, w) = \max_{v \in V} r(v, w(v))$ in the egalitarian case.

Observe how $w(V)$ in the above is the smallest committee that is consistent with w being an assignment of its members to the voters. Since enlarging a committee can not possibly make the total dissatisfaction any worse, unless w itself is changed, it is natural to assume that, from now on, whenever a function w is available, then the corresponding committee is always $w(V)$, implicitly.

Definition 2.7. The *Chamberlin–Courant* multi-winner rule (CC) takes a set of voters V, their preferences P and the number of representatives to be elected k as input and outputs an optimal k-assignment w_{opt} of voters to representatives that minimizes the total dissatisfaction $\Phi(V, w)$.

Depending on whether the utilitarian or the egalitarian definitions of total dissatisfactions have been used, we distinguish between *utilitarian CC* and *egalitarian CC*. Note that the original rule β-CC introduced by Chamberlin and Courant [7] is an instance of utilitarian CC with *Borda dissatisfactions*: $r(v, c) = pos_v(c)$.

Next, we define what it means for preferences to be single-crossing, in both the classical and generalized senses:

Definition 2.8. A preference profile is said to be *single-crossing* (in the classical sense) if there is a total order $\lhd$ over V such that for any three voters a, b, c such that $a \lhd b \lhd c$ and candidates c_0, c_1 such that $c_0 \succ_a c_1$ and $c_0 \succ_c c_1$, it necessarily also holds that $c_0 \succ_b c_1$.

Intuitively, this means that voters can be ordered into a list such that, as we sweep through it from left to right, the relative order of every pair of candidates changes at most once (i.e. there is at most one point at which preference for one candidate over the other changes, and hence the name single-crossing). Figure 2 illustrates this notion.

Definition 2.9. A finite[2] undirected graph G is called a *median graph* if for any three vertices a, b, c there is a unique vertex $m(a, b, c)$, called the *median* of a, b, c, which is simultaneously on one $a - b$, one $b - c$ and one $c - a$ shortest path.

Definition 2.10. A preference profile is said to be *single-crossing with respect to a median graph* having its n voters as vertices if for any two voters s, t and all $s - t$ shortest paths X the profile induced by the voters in X is single-crossing in the classical sense with respect to the natural order induced by X.

Intuitively, this means that whenever we walk in order through the voters of an $s - t$ shortest path, the relative order of every pair of candidates changes at most once. Note that if the graph happens to be a path this becomes classical single-crossingness.

[2]In this work we are not concerned with infinite structures.

Next, we define two well-known types of undirected graphs (and some related notions). Such graphs turn out to always be median graphs:

Definition 2.11. A *tree* is an undirected connected acyclic graph.

Definition 2.12. A *rooted tree* is a tree with a designated root vertex. We think of its edges as being oriented away from the root. With this in mind, we define:

- $parent(v)$ to be the parent of vertex v.
- n_v to be the number of children of vertex v.
- $child[v]$ to be the children of v, in the form of a 1-indexed array: $child[v, 1]$, $child[v, 2]$, ..., $child[v, n_v]$.

Definition 2.13. With respect to a rooted tree, we call edges of the form $(v, parent(v))$ *parent links* and edges of the form $(v, child[v, i])$ *child links*.

Definition 2.14. Let T_v denote the subtree of T induced by all vertices reachable from v by following child links only. Similarly, let $T_{v,i}$ for $1 \leq i \leq n_v + 1$ be the subtree of T induced by all vertices reachable from v by following child links only, other than those from v to its first $i - 1$ children.

Observe that $T_{v,1} = T_v$ and that T_{v,n_v+1} is just the singleton vertex v.

Definition 2.15. Let $|T|$ denote the number of nodes of a tree T.

Definition 2.16. Given a rooted tree and two vertices v_1, v_2, we call their *lowest common ancestor* (LCA) the unique node on the $v_1 - v_2$ simple path that is closest to the root of the tree.

Definition 2.17. A *d-dimensional grid graph (d-grid)* has vertex set $V = [n_1] \times [n_2] \times \ldots \times [n_d]$ and edges between $(a_1, a_2, \ldots, a_d)$ and $(b_1, b_2, \ldots, b_d)$ if and only if $\sum_{i=1}^{d} |a_i - b_i| = 1$. A 2-grid is simply called a *grid* or *rectangle*.

By extension, we also call all graphs isomorphic with this construction d-grids.

Definition 2.18. A *subgrid* of a d-grid is a subgraph induced by a subset $V' \subseteq V$ of the form $V' = [a_1 : b_1] \times [a_2 : b_2] \times \ldots \times [a_d : b_d]$, where (a_1, b_1), (a_2, b_2), ..., (a_d, b_d) are such that $1 \leq a_i \leq b_i \leq n_i$ for all $1 \leq i \leq d$. When $d = 2$ we also call subgrids *subrectangles*.

Lemma 2.19. *Paths, trees and d-grids are all median graphs.*

Proof. Either by directly checking the median condition, or by induction and Mulder's Convex Expansion Theorem [24]. □

We continue by introducing some of the terminology which will be needed in Chapter 3:

Definition 2.20. A *directed acyclic graph* (DAG) is a finite oriented graph whose vertices can be totally ordered so that all arcs have their head coming before their tail in the ordering. Unless stated otherwise, all DAGs we will consider have a subset of the integers as their vertices and are ordered with respect to the natural ordering $<$. A DAG is said to be weighted if its edges are given real values by a function w.

Definition 2.21. A weighted DAG satisfies the *concave Monge* property if for any four vertices a, b, c, d such that $a < b < c < d$ it holds that $w(a, c) + w(b, d) \leq w(a, d) + w(b, c)$. We refer to the weight function/matrix w itself as being concave Monge in this case.

We conclude with a remark about the way we will use Definition 2.5.

Remark 2.22. Observe that the inequality $r(v, c_0) \leq r(v, c_0)$ in Definition 2.5 is non-strict, meaning that a voter is allowed to be equally dissatisfied with two different candidates. While this is convenient in general, it makes some results about optimal k-assignments weaker; i.e. statements of the form "all optimal assignments satisfy property P" should be replaced with "there is an optimal assignment satisfying property P". It is not hard to see that, by varying the r values by infinitesimal amounts and applying the stronger results that hold for distinct r values, the weaker results also follow. Therefore, for some proofs we will assume that each voter's r values are distinct and prove the stronger version, instead.

3. Improved Algorithms for Classical Single-Crossing CC

Skowron et al. [30] gave a Dynamic Programming (DP) algorithm for solving CC in time $O(n^2mk)$ for classical single-crossing electorates. First, we improve this algorithm to $O(nmk)$. Then, we introduce the k-Link Path Problem and show how utilitarian CC reduces to its concave Monge case, allowing us to deploy techniques specific to this problem. This leads to an improved time bound of $nm2^{O(\sqrt{\log k \log\log n})}$ for $k = \Omega(\log n)$. We end by pointing out how we also get $O(nm \log n \log(nm))$ for the egalitarian case.

3.1 A $O(nmk)$ Algorithm for Classical Single-Crossing CC

Assume that $V = [n]$, $C = [m]$ and that the preferences are single-crossing with respect to the natural ordering of the integers. Additionally, without loss of generality, assume that $1 \succ_1 2 \succ_1 \ldots \succ_1 m$. While our algorithm applies to both utilitarian and egalitarian CC, to keep notation simple, we will only present the former. Additionally, as per Remark 2.22, we assume that the r values of each voter are distinct.

The following Lemma is instrumental to both our algorithm and the one of Skowron et al:

Lemma 3.1. *In any optimal assignment w_{opt} and for any two candidates v, v' such that $v < v'$ it holds that $w_{opt}(v) \leq w_{opt}(v')$.*

We prove a more general version of this statement in the next chapter. It has the useful implication that any optimal assignment w_{opt} partitions the path into contiguous subsequences.

We will proceed by DP. We define the subproblems to be of the form $dp_f[v, \ell, c]$, representing the minimal possible dissatisfaction of voters $[v : n]$ when picking an ℓ-member committee out of candidates $[c : m]$, with the extra condition that if $f \in \{0, 1\}$ has value 1, then voter v is required to be represented by candidate c. We stress the fact that these subproblems are only defined for $1 \leq v \leq n$, $1 \leq \ell \leq k$, $1 \leq c \leq m$ and are considered to have value ∞ otherwise. We now present the base cases and recurrence relations that we will use to evaluate the DP:

Lemma 3.2. *The following hold:*

$$dp_1[v, \ell, c] = \begin{cases} r(v, c), & \text{if } v = n \\ r(v, c) + \min(dp_1[v+1, \ell, c], \\ \qquad dp_0[v+1, \ell-1, c+1]), & \text{if } v < n \end{cases} \tag{3.1.1}$$

$$dp_0[v, \ell, c] = \min(dp_1[v, \ell, c], dp_0[v, \ell, c+1]). \tag{3.1.2}$$

The second equality might remind the reader of partial sums/maxima.

Proof. (3.1.1) The first branch is immediate by definition. For the second branch, note that, other than the cost $r(v, c)$ added by v being represented by c, we need to case split on which candidate represents voter $v + 1$:

- If $v + 1$ is represented by c, then w_{opt} optimally represents voters $[v + 1 : n]$ by an ℓ-member committee chosen from candidates $[c : m]$. Therefore, in this case the optimal cost is $r(v, c) + dp_1[v + 1, \ell, c]$.
- If $v + 1$ is represented by $c' > c$, then w_{opt} optimally represents voters $[v + 1 : n]$ by a committee of size $\ell - 1$ chosen from candidates $[c' : m]$ (by Lemma 3.1). From this we know that candidate c will not represent any voters past v, so the optimal cost in this case is $r(v, c) + dp_0[v + 1, \ell - 1, c + 1]$.

(3.1.2) The optimal ℓ-member committee for voters $[v : n]$ picked out of candidates $[c : m]$ is either one where voter v is represented by c or one where candidate c is not part of the committee, as a result of Lemma 3.1. Since the optimal dissatisfaction is $dp_1[v, \ell, c]$ in the former case and $dp_0[v, \ell, c + 1]$ in the latter, the result follows. □

With this in mind, an $O(nmk)$ algorithm for classical single-crossing CC can now easily be given.

Theorem 3.3. *Classical single-crossing CC can be solved in time $O(nmk)$.*

Proof. Our goal is to compute $dp_0[1, k, 1]$. In order to do this, we solve the subproblems in decreasing order of v, breaking ties by decreasing f, and further ties by decreasing c. Since there are $O(nmk)$ subproblems, each solvable in constant time via Lemma 3.2, we get the conclusion. Using standard techniques we can walk through the DP table to also construct an explicit optimal assignment w_{opt} without an increase in time complexity—in the future we will not mention this fact when reconstructing a solution from the DP table is straightforward. □

To conclude we note that the only essential difference between our algorithm and that of Skowron et al. is in using two types of subproblems rather than one: dp_0 and dp_1. Indeed, it turns out that dp_0 can be computed directly, without the need for dp_1, but this adds an extra linear factor to the complexity (see Skowron et al. [30] for the details).

3.2 The k-Link Path Problem and Monge Matrices

In the following sections we introduce a well-studied problem, the *DAG k-Link Path Problem*, and show how classical single-crossing utilitarian CC[1] reduces to it. We briefly discuss *concave Monge* weight matrices and then prove that matrices arising from our reduction satisfy this property. We conclude by discussing the algorithmic implications of this result.

Definition 3.4. Given a DAG with edges (i, j) of weight $w(i, j)$ for all $i < j$, the *k-Link Path Problem* (LPP) asks for a minimum total weight path starting at s, ending at t and consisting of exactly k edges, where s and t denote the minimal/maximal elements in the ordering of the vertices.

[1]For the remainder of this chapter we will often refer to this simply as CC.

To reduce CC to LPP we start with an instance of CC. The corresponding DAG G will have vertex set $[0:n]$ and w defined as follows:

$$w(i,j) = \min_{c \in C} \left(r(i+1,c) + \ldots + r(j,c) \right) \tag{3.2.1}$$

In other words, $w(i,j)$ represents the minimal total dissatisfaction to represent voters $[i+1:j]$ using a single candidate c. Denote a c for which minimum is attained by $cand(i,j)$.

Theorem 3.5. *The minimum cost of a k-link path in G is equal to the minimum total dissatisfaction for an instance of CC.*

Proof. Let $P = 0 \to i_1 \to \ldots \to i_{k-1} \to n$ be a minimum cost k-link path in G. Then, P induces an assignment of candidates to voters, assigning candidate $cand(i,j)$ to voters $[i+1:j]$. This assignment has a total dissatisfaction equal to the total weight of P.

Conversely, let w_{opt} be an optimal assignment of candidates to voters. We construct a path P in G, as follows. By Lemma 3.1 we know that w_{opt} partitions the voters into contiguous subsequences. Let $[\ell : r]$ be any maximal contiguous subsequence of voters represented by the same candidate in w_{opt}, then, we say that edge $(\ell - 1) \to r$ belongs to P. Note that the set of all edges defined by this procedure indeed forms a k-link path from 1 to n, whose total weight is at most equal to $\Phi(V, w_{opt})$. This is because w_{opt} gives out an explicit assignment of candidates to subsequences, while the path P actually picks the best candidate for each subsequence. If we now use the first part of the proof again on P we get a contradiction unless the total weight of P exactly equals $\Phi(V, w_{opt})$. □

In fact, the proof of the last theorem tells us precisely how solving LPP can be used to solve classical CC, fact which we note in the following proposition:

Proposition 3.6. *If P is a minimal weight path in G, then the assignment w_{opt} induced by P is a minimal dissatisfaction solution for CC.*

This being said, any fast algorithm for LPP can now successfully be deployed to solve CC with better time bounds. In order to do this, we will now show that w defined in (3.2.1) above has the concave Monge property. As preparation, we first state and prove a classical lemma:

Lemma 3.7. *The following two conditions are equivalent:*

1. *w has the concave Monge property.*
2. *For all vertices i, j such that $0 < i+1 < j < n$ it holds that $w(i,j) + w(i+1,j+1) \leq w(i,j+1) + w(i+1,j)$.*

Proof. Note that (2) is an edge case of (1), so one direction is trivial. For the other direction, assume (2) holds and consider four arbitrary vertices a, b, c, d such that $a < b < c < d$. We then need to show that $w(a,c) + w(b,d) \leq w(a,d) + w(b,c)$. In order to do this, in (2) we substitute i with each of $[a : b-1]$ and j with each of $[c : d-1]$. Summing up these inequalities and cancelling like terms we get the conclusion. □

From now on, we will use these two definitions interchangeably. We will also call a CC instance *concave Monge* if the corresponding weighting function w has the concave Monge property.

We now prove two lemmas about non-concave Monge instances of CC. These will help us restrict the set of candidate counterexamples to only a finite set, which will then be checked on a one-by-one basis, ultimately showing that no such instances actually exist.

Lemma 3.8. *Assume a non-concave Monge instance of single-crossing CC exists, then there is a non-concave Monge instance with three voters.*

Proof. If $n \leq 3$, then we are done by cloning voters and setting the new dissatisfactions to zero, so assume $n \geq 4$. Without loss of generality, assume that the $(i,\ j)$ pair which does not satisfy the Monge condition is $(0,\ n-1)$, as otherwise we could just remove all voters before i and all voters after j. The only quantities that matter to us now are $w(0, n-1)$, $w(0, n)$, $w(1, n-1)$ and $w(1, n)$. Observe how in these four quantities voters $[2 : n-1]$ always come together, as a whole, contributing exactly $r(2, c) + r(3, c) + \ldots + r(n-1, c)$ to the sum each time.

The idea will now be to replace voters $2, 3, \ldots, n-1$ by a single aggregated voter x, whose dissatisfaction function $r(x, c)$ is given by $r(2, c)+r(3, c)+\ldots+r(n-1, c)$. We let x's preferences be defined by an arbitrary total order $\succ_x$ such that whenever $c \succ_x c'$ it also holds that $r(x, c) \leq r(x, c')$. We now claim that the so formed 3-voter profile is single-crossing with respect to the order $1 \triangleleft x \triangleleft n$. In order to check this, we need to show that whenever it holds that $c \succ_1 c'$ and $c \succ_n c'$ it also holds that $c \succ_x c'$. Assume that the former two hold, therefore, since the initial profile was single-crossing, we get that voters $[2 : n-1]$ all prefer c to c', implying that

$$r(2, c) + r(3, c) + \ldots + r(n-1, c) < r(2, c') + r(3, c') + \ldots + r(n-1, c')$$

so, by definition, $r(x, c) < r(x, c')$, and hence the conclusion.

Therefore, the CC instance formed by 1, x and n is also non-concave Monge. □

Lemma 3.9. *Assume a non-concave Monge instance of single-crossing CC with three voters exists, then there is a non-concave Monge instance with three voters and four candidates.*

Proof. The instance formed by removing all candidates except for $cand(0, 2)$, $cand(0, 3)$, $cand(1, 2)$ and $cand(1, 3)$ is a non-concave Monge instance of CC. If less than four candidates remain, then we add additional candidates which are last in the preferences of all voters and have very large associated dissatisfactions. □

We are now in a position of having reduced the search space of possible non-concave Monge instances of CC to 3-voter profiles with four candidates. Observe that the number of 3-voter single-crossing profiles is finite (and relatively small, empirically 151), so our proof can very well be by exhaustion over all cases. In the next section we do just that.

3.3 An LP Proof of Monge-Concavity for Classical Single-Crossing Utilitarian CC

This section constitutes the proof of Monge-Concavity for CC. In Lemma 3.9 we showed that if Monge-Concavity is ever violated, then a counterexample with three voters and four candidates

necessarily exists. Therefore, we can consider without loss of generality only CC instances of this small size. We begin by putting together some working premises, to which we will refer by their number (e.g. (1) for the first):

1. There are three voters: $1, 2, 3$ and four candidates: $1, 2, 3$ and 4.
2. The preferences of the voters are single-crossing with respect to the ordering $1 \lhd 2 \lhd 3$.
3. Without loss of generality, voter 1 has identity preferences $\sigma_1(i) = id(i) = i$:

$$1 \succ_1 2 \succ_1 3 \succ_1 4$$

4. Voters 2 and 3 have the preferences one would expect from equation (2.0.1):

$$\sigma_2(1) \succ_2 \sigma_2(2) \succ_2 \sigma_2(3) \succ_2 \sigma_2(4)$$
$$\sigma_3(1) \succ_3 \sigma_3(2) \succ_3 \sigma_3(3) \succ_3 \sigma_3(4)$$

5. Misrepresentations are monotonic, as in Definition 2.5. Additionally, we can assume without loss of generality that the most preferred candidate of each voter has a misrepresentation of 0:

$$0 = r(1,1) \leq r(1,2) \leq r(1,3) \leq r(1,4)$$
$$0 = r(2,\sigma_2(1)) \leq r(2,\sigma_2(2)) \leq r(2,\sigma_2(3)) \leq r(2,\sigma_2(4))$$
$$0 = r(3,\sigma_3(1)) \leq r(3,\sigma_3(2)) \leq r(3,\sigma_3(3)) \leq r(3,\sigma_3(4))$$

Before proceeding any further, we first prove a lemma which identifies non-concavity with a condition which is easier to check algorithmically:

Lemma 3.10. *The following two statements are equivalent (in our setting):*

1. *Monge-Concavity fails.*
2. *There is a candidate $c'' \in [4]$ such that for all candidates $c, c' \in [4]$ it holds that $r(1,c) + r(2,c) + r(2,c') + r(3,c') > r(1,c'') + r(2,c'') + r(3,c'')$.*

Proof.

$$
\begin{aligned}
& w(0,2) + w(1,3) > w(0,3) + w(1,2) \\
\overset{\text{def}}{\iff} & \\
& \min_{c\in[4]}(r(1,c) + r(2,c)) + \min_{c\in[4]}(r(2,c) + r(3,c)) > \\
& \min_{c\in[4]}(r(1,c) + r(2,c) + r(3,c)) + \min_{c\in[4]}(r(2,c)) \\
\overset{(5)}{\iff} & \\
& \min_{c\in[4]}(r(1,c) + r(2,c)) + \min_{c\in[4]}(r(2,c) + r(3,c)) > \min_{c\in[4]}(r(1,c) + r(2,c) + r(3,c)) \\
\iff & \\
& \text{For all } c, c' \in [4] \text{ it holds that} \\
& r(1,c) + r(2,c) + r(2,c') + r(3,c') > \min_{c''\in[4]}(r(1,c'') + r(2,c'') + r(3,c'')) \\
\iff & \\
& \text{There is a } c'' \in [4] \text{ such that for all } c, c' \in [4] \text{ it holds that} \\
& r(1,c) + r(2,c) + r(2,c') + r(3,c') > r(1,c'') + r(2,c'') + r(3,c'')
\end{aligned}
$$

□

These being said, to prove Monge-Concavity we proceed by listing all triples $(\sigma_2, \sigma_3, c'')$ such that the profile $P = (id, \sigma_2, \sigma_3)$ is single-crossing and c'' is as in Lemma 3.10 above. For each such triple, we will formulate a linear program (LP) that is feasible if and only if Monge-Concavity fails for some choice of misrepresentations r respecting preferences P. Specifically, each LP will enforce that premise (5) is satisfied and that for all $c, c' \in [4]$ condition (2) in Lemma 3.10 holds. We have formulated these LPs using the C++ API of the Z3 Theorem Prover and concluded that none of them, indeed, are satisfiable. The code performing this check is available in the Appendix.

Remark 3.11. One might argue that our LPs' feasible regions are not closed (due to the strict inequality in the non-concavity condition) and so we might not be able to apply the standard algorithms. However, all constants appearing in the LPs are rational, and so the hyperplanes bounding the feasible regions are rational, too. Therefore, if a counterexample exists, then a rational one does too. Since all inequalities are also homogeneous, by multiplying all r values by the product of the denominators, we get that an integer solution also exists in this case. This means that

$$r(1,c) + r(2,c) + r(2,c') + r(3,c') > r(1,c'') + r(2,c'') + r(3,c'')$$

can be replaced by

$$r(1,c) + r(2,c) + r(2,c') + r(3,c') \geq 1 + r(1,c'') + r(2,c'') + r(3,c'')$$

while preserving satisfiability.

3.4 Implications of the Reduction

Having reduced classical single-crossing utilitarian CC to Concave Monge LPP, any fast algorithm for the latter translates into one for the former, slowed down by a factor of $O(m)$ required for computing edge weights.[2] When $k = O(\log n)$, it does not seem like any of the techniques outlined next can be used to improve on our $O(nmk)$ algorithm from section 3.1. However, when $k = \Omega(\log n)$, a number of attractive results are possible. First, if we assume that individual dissatisfactions are integers bounded by some positive integer U (e.g. $U = m$ for β-CC), then the weakly-polynomial algorithm of Bein, Larmore and Park [3] and Aggrawal, Schieber and Tokuyama [1] leads to an $O(nm \log(nU))$ algorithm for CC. If, instead, we use the strongly-polynomial time algorithm of Schieber [28], we get a runtime of $nm2^{O\left(\sqrt{\log k \log\log n}\right)}$, which can be seen to improve on our earlier bound of $O(nmk)$ for $k = \Omega(\log n)$. To conclude, we further remark that one can solve egalitarian CC by binary searching for the optimal dissatisfaction and then running the $O(nm \log(nU))$ algorithm for the utilitarian case with individual dissatisfactions from the set $\{0, 1\}$. This leads to a $O(nm \log n \log U)$ algorithm for the egalitarian case. This can be made strongly-polynomial by restricting the binary search to values which are a voter's dissatisfaction, for a time bound of $O(nm \log n \log(nm))$.

[2] Computing all edge weights in advance would be too expensive, so we only compute weights when needed by the algorithm.

4. A Polynomial-Time Algorithm for CC on Tree Single-Crossing Preferences

Clearwater, Puppe and Slinko [8] proposed an algorithm for computing winners of tree single-crossing CC. Unfortunately, the analysis of the proposed algorithm turns out to be incorrect, leading to a worst case exponential-time algorithm.[1] We hereby give the first correct polynomial-time algorithm for this problem, running in time $O(nmk)$.

4.1 A Simplifying Reduction, Introducing CCTP

In this section we define the *Chamberlin–Courant Tree Partition problem* (CCTP) and show how CC on tree single-crossing preferences[2] reduces to CCTP. While our algorithm applies to both utilitarian and egalitarian CC, to keep notation simple, we will only present the former. Additionally, as per Remark 2.22, we assume that the r values of each voter are distinct.

For simplicity, assume $C = [m]$ and that the voters $V = [n]$ are the vertices of a tree T rooted in voter 1, with respect to which the preferences are single-crossing. Additionally, without loss of generality, assume $1 \succ_1 2 \succ_1 3 \succ_1 \ldots \succ_1 m$. We begin by presenting some properties of optimal k-assignments w_{opt}:

Lemma 4.1. *Let w_{opt} be an optimal k-assignment, then for any candidate c it holds that the inverse image $w_{opt}^{-1}(c)$ defines a subtree of T.*

In other words, w_{opt} induces a partition of T into at most k subtrees, each subtree representing a group of voters with the same representative.

Proof. Let T' be the smallest subtree of T which contains voters $w_{opt}^{-1}(c)$. Assume for a contradiction that there is a voter v in T' such that $w_{opt}(v) = c' \neq c$. Then, there are 2 voters $v_0, v_1 \in T' \cap w_{opt}^{-1}(c)$ for which the unique simple $v_0 - v_1$ path P contains v. Since w_{opt} is optimal, it means that $c \succ_{v_0} c'$ and $c \succ_{v_1} c'$, but $c' \succ_v c$, contradicting single-crossingness over P.

The existence of v_0 and v_1 follows from the fact that T' is in fact the union of all simple paths between any 2 vertices in $w_{opt}^{-1}(c)$. □

[1] Like ours, their algorithm proceeds by DP. However, the recurrence essentially picks a subtree of the form T_v and removes it from the tree, recursing to $T \setminus T_v$. It is not hard to see that on a star tree instance this leads to all subproblems of the form $T \setminus L$ being considered, where L is any set of leaves not containing the root. Since the tree has $n - 1$ leaves, the algorithm is exponential-time. The error has been pointed out to the authors via email and they agreed with our argument.

[2] Which we will usually refer to as simply CC for the remainder of the chapter.

Lemma 4.2. *Let w_{opt} be any optimal k-assignment and P any simple path starting at voter 1, then w_{opt} is non-decreasing along P.*

In other words, as we start from voter 1 and walk down the tree using child links only, the value of the assignment function w_{opt} can only ever increase. In particular, this means that voters with higher numbered representatives need to be further down in the tree. Finally, note that when T is a path, this is precisely Lemma 3.1.

Proof. Assume for a contradiction there was a path P starting at voter 1 and two voters v_0, v_1 on P such that v_0 comes before v_1 on P and $w_{opt}(v_0) = c_0$, $w_{opt}(v_1) = c_1$ with $c_0 > c_1$. This means that $c_1 \succ_1 c_0$, $c_0 \succ_{v_0} c_1$ and $c_1 \succ_{v_1} c_0$, contradicting single-crossingness over P. □

The key ingredient to our polynomial-time algorithm will be noting that CC is in some sense equivalent to the following related problem:

Definition 4.3. The *Chamberlin–Courant Tree Partition* (CCTP) problem takes a set of voters V, a profile P which is single-crossing with respect to a tree T and a number k as input and outputs a partition F of T into at most k subtrees, together with a k-assignment w, mapping voters in the same subtree of the partition to the same representative. The goal is, once again, to minimize $\Phi(V, w)$.

The next theorem provides us with a sense in which CC and CCTP are equivalent.

Theorem 4.4. *The minimal total dissatisfaction is the same for both CC and CCTP.*

Proof. Let w_{opt} be an optimal solution to CC. By Lemma 4.1 we obtain that w_{opt} induces a natural partition F_{opt} of T into at most k subtrees, since w_{opt} is a k-assignment. Therefore, (F_{opt}, w_{opt}) is a solution to CCTP of the same dissatisfaction $\Phi(V, w_{opt})$ as for CC.

Conversely, let (F_{opt}, w_{opt}) be an optimal solution to CCTP. Then, as $|F_{opt}| \leq k$, it follows that w_{opt} is a k-assignment, having the same dissatisfaction $\Phi(V, w_{opt})$ for both CCTP and CC. □

In fact, the last part of the proof tells us precisely how solving CCTP can be used to solve CC. Formally, we have the following proposition:

Proposition 4.5. *If (F_{opt}, w_{opt}) is an optimal solution to CCTP, then w_{opt} is an optimal solution to CC.*

Proposition 4.5 and Lemma 4.2 together imply the following useful corollary:

Corollary 4.6. If (F_{opt}, w_{opt}) is an optimal solution to CCTP, then w_{opt} is non-decreasing as we walk down the tree using child links only.

While this reduction from CC to CCTP may, at first, seem of no practical use, the favor that we have done ourselves is that it is much easier to control the number of subtrees in a partition than the number of elected candidates, because voters represented by any given candidate might not form a subtree in general, the fact that this never happens in optimal solutions proving key to the reduction.

4.2 A DP Algorithm for CCTP

In this section we present an algorithm which solves CCTP in time $O(nmk^2)$. In the next section we show that, with more careful analysis, the bound on its running time can be improved to $O(nmk)$.

Without loss of generality, assume $k \leq n$. In order to proceed by DP, we start by defining three types of subproblems:

1. $dp_0[v, \ell, c]$ = the minimal dissatisfaction of voters in T_v which can be achieved by partitioning T_v into ℓ subtrees using only a subset of candidates $[c : m]$ as representatives. This is only defined when $1 \leq \ell \leq \min(|T_v|, k)$.
2. $dp_1[v, \ell, c]$ = the minimal dissatisfaction of voters in T_v which can be achieved by partitioning T_v into ℓ subtrees using only a subset of candidates $[c : m]$ as representatives and having voter v be represented by candidate c. This is only defined when $1 \leq \ell \leq \min(|T_v|, k)$.
3. $dp_2[v, i, \ell, c]$ = the minimal dissatisfaction of voters in $T_{v,i}$ which can be achieved by partitioning $T_{v,i}$ into ℓ subtrees using only a subset of candidates $[c : m]$ as representatives and having voter v be represented by candidate c. This is only defined when $1 \leq i \leq n_v + 1$ and $1 \leq \ell \leq \min(k, |T_{v,i}|)$.

Of course, these quantities only make sense when $1 \leq v \leq n, 1 \leq \ell \leq k$ and $1 \leq c \leq m$. We say that all undefined subproblems take value ∞.

It is not hard to see that $dp_0[1, k, 1]$ is the overall minimal dissatisfaction that we ultimately aim to compute. This is because dp_0, dp_1 and dp_2 are all non-decreasing in ℓ, as a partition into ℓ subtrees can always be turned into a partition into $\ell + 1$ subtrees of no greater cost by picking an arbitrary subtree of size at least 2 and subdividing it.

We now present a number of equalities involving dp_0, dp_1 and dp_2 which will constitute the base cases and recurrence relations that will be used to compute the answer for each subproblem. First, we note several simpler facts, as follows:

Lemma 4.7. *The following hold:*

$$dp_2[v, n_v + 1, \ell, c] = r(v, c) \tag{4.2.1}$$

$$dp_1[v, \ell, c] = dp_2[v, 1, \ell, c] \tag{4.2.2}$$

$$dp_0[v, \ell, c] = \min(dp_1[v, \ell, c], dp_0[v, \ell, c + 1]) \tag{4.2.3}$$

The last equality might remind the reader of partial sums/maxima.

Proof. All three follow almost directly from definitions:
(4.2.1) Note that $T_{v,n_v+1} = \{v\}$ and that the unique partition of $\{v\}$ has cost $r(v, c)$.
(4.2.2) By definition.
(4.2.3) The left-hand side is the minimal dissatisfaction to partition T_v into ℓ subtrees such that v is represented by some candidate in $[c : m]$. We can partition this set into $\{c\}$ and $[c + 1 : m]$, which is precisely what the two subproblems on the right-hand side stand for, by definition. □

We now prove a more subtle fact, the key to making the whole approach work:

Lemma 4.8. *For convenience, let v' be shorthand for child$[v, i]$ and define the following two quantities:*

$$DIFF = \min_{\substack{1 \le \ell' \le \min(\ell, |T_{v'}|) \\ 1 \le \ell - \ell' \le \min(\ell, |T_{v,i+1}|)}} (dp_0[v', \ell', c+1] + dp_2[v, i+1, \ell - \ell', c]) \tag{4.2.4}$$

$$SAME = \min_{\substack{1 \le \ell' \le \min(\ell, |T_{v'}|) \\ 1 \le \ell - \ell' + 1 \le \min(\ell, |T_{v,i+1}|)}} (dp_1[v', \ell', c] + dp_2[v, i+1, \ell - \ell' + 1, c]) \tag{4.2.5}$$

Then, it holds that $dp_2[v, i, \ell, c] = \min(DIFF, SAME)$.

Proof. Throughout this proof we will make implicit use of Lemma 4.2 to justify always having candidates with a higher index be further down in the tree.

Let (F_{opt}, w_{opt}) be a minimal dissatisfaction partition of $T_{v,i}$ into ℓ subtrees such that voter v is assigned candidate c. There are two cases to consider:

(4.2.4) Voter v' is represented by some candidate $c' > c$. Then, observe that each subtree in F_{opt} is either fully contained in $T_{v'}$ or in $T_{v,i+1}$, so there is a number ℓ' such that F_{opt} partitions $T_{v'}$ into ℓ' subtrees and $T_{v,i+1}$ into $\ell - \ell'$ subtrees. Knowing this, in order to get the best dissatisfaction value we need to take the minimum over all possible values of ℓ'. For a fixed ℓ' we need to choose an optimal partition of $T_{v'}$ into ℓ' subtrees using only candidates $[c+1 : m]$ and an optimal partition of $T_{v,i+1}$ into $\ell - \ell'$ subtrees such that v is represented by c. By definition, the optimal value of the former is precisely $dp_0[v', \ell', c+1]$, and that of the latter is precisely $dp_2[v, i+1, \ell - \ell', c]$.

(4.2.5) Voter v' is represented by candidate c. In this case the only added complication is that the subtree in F_{opt} which contains v partly resides in both $T_{v'}$ and $T_{v,i+1}$. Therefore, this time there is a number ℓ' such that F_{opt} partitions $T_{v'}$ into ℓ' subtrees and $T_{v,i+1}$ into $\ell - \ell' + 1$ subtrees. Once again, we take the minimum over all possible values of ℓ'. For a fixed ℓ' we choose an optimal partition of $T_{v'}$ into ℓ' subtrees such that v' is represented by c and an optimal partition of $T_{v,i+1}$ into $\ell - \ell' + 1$ subtrees such that v is also represented by c. By definition, the optimal values for these subproblems are $dp_1[v', \ell', c]$ and $dp_2[v, i+1, \ell - \ell' + 1, c]$, respectively.

Consequently, *SAME* and *DIFF* are the minimal dissatisfactions that can be achieved in each case separately. Therefore, the conclusion follows. There is one more detail required for completeness: we must enforce that all relevant subproblems are well-defined, meaning that $1 \le \ell' \le \min(\ell, |T_{v'}|)$ and $1 \le \ell - \ell' + 1 \le \min(\ell, |T_{v,i+1}|)$, by essentially immediate reference to the constraints in the definitions of dp_0, dp_1 and dp_2. Note that the $+1$ in $\ell - \ell' + 1$ shall be included for *SAME* and omitted for *DIFF*. □

Importantly, one should note that the seemingly complex constraints on ℓ' under the min operators in equations (4.2.4) and (4.2.5) define ranges of integers. Also, note that these recurrences allow for the same candidate to be the representative of multiple subtrees of the partition, but this will never be the case in an optimal solution, as proven in the previous section.

Armed with these tools, we now prove the main theorem of the section:

Theorem 4.9. *CCTP can be solved in time $O(nmk^2)$.*

Proof. As noted above, our ultimate goal is to compute $dp_0[1, k, 1]$. Computing this quantity leads to the need to solve other subproblems, as with any DP algorithm. It is now easiest to proceed by solving all subproblems defined at the beginning of the section.

Observe that the subproblems can be partially ordered as follows:

1. Let v, v' be any two voters such that $T_{v'} \subset T_v$, then all subproblems involving v come after all subproblems involving v'.
2. Let v be any voter, then all subproblems of the forms $dp_0[v, -, -]$ and $dp_1[v, -, -]$ come after all subproblems of the form $dp_2[v, -, -, -]$.
3. Let v be any voter and let i, i' be such that $1 \leq i < i' \leq n_v + 1$, then all subproblems of the form $dp_2[v, i, -, -]$ come after all subproblems of the form $dp_2[v, i', -, -]$.
4. Let v be any voter and let c, c' be any two candidates such that $1 \leq c < c' \leq m$. Additionally, let $f, f' \in \{0, 1\}$, then all subproblems of the form $dp_f[v, -, c]$ come after all subproblems of the form $dp_{f'}[v, -, c']$.

It is not difficult to check that this is, indeed, a valid partial order (i.e. it is acyclic) and that whenever a subproblem is solved in terms of other subproblems (as in Lemma 4.7 and Lemma 4.8), those subproblems come before it in the partial order.

It is easy to construct one such order explicitly via a single depth first traversal of T starting from voter 1. Then, we can solve the subproblems in this order by applying the equations given in Lemma 4.7 and Lemma 4.8.

For the time complexity, note that:

1. There are $O(nmk)$ subproblems of the form $dp_0[-, -, -]$ and $dp_1[-, -, -]$, each requiring constant time to solve.
2. There are $O(nmk)$ subproblems of the form $dp_2[-, -, -, -]$. This is because pairs of the form (v, i) such that $1 \leq i \leq n_v$ correspond to edges of the tree and there are precisely $O(n)$ of them. Each of these subproblems requires time at most $O(k)$ to solve.
3. The ordering of the subproblems can be computed in time $O(nmk)$.

Altogether, we get a polynomial time bound of $O(nmk^2)$. □

4.3 A Better Time Complexity Bound

It turns out that this time bound of $O(nmk^2)$ is not tight. Rather, we will see that a better bound of $O(nmk)$ holds for the same algorithm.

To begin, observe that time complexity in the proof of Theorem 4.9 is dominated by solving subproblems of the form $dp_2[-, -, -, -]$, so all we need to do is to show that solving all of them using the recurrences given in Lemma 4.8 takes time $O(nmk)$.

Lemma 4.10. *Let v be any voter, i be such that $1 \leq i \leq n_v$ and c be such that $1 \leq c \leq m$. Additionally, let v' denote child$[v, i]$. Then, solving all subproblems of the form $dp_2[v, i, -, c]$ using Lemma 4.8 takes time $O(\min(k, |T_{v'}|) \cdot \min(k, |T_{v,i+1}|))$. Importantly, here and in all subsequent results, solving a subproblem will exclude the time required to solve all subproblems that it depends on.*

Proof. We first look at the time required to compute *DIFF* from Lemma 4.8 for all $1 \leq \ell \leq \min(k, |T_{v,i}|)$. For each such ℓ define a set

$$M_\ell = \{(\ell, \ell') : 1 \leq \ell' \leq \min(\ell, |T_{v'}|) \text{ and } 1 \leq \ell - \ell' \leq \min(\ell, |T_{v,i+1}|)\}$$

consisting of those pairs (ℓ, ℓ') which will be considered at some point in the expression for *DIFF* when solving all subproblems of the form $dp_2[v, i, -, c]$. Clearly, the time it takes to compute *DIFF* for all cases is bounded by the total size of these sets. Now, apply the bijective map $(\ell, \ell') \mapsto (\ell - \ell', \ell')$ to each M_ℓ to get a new collection of sets

$$M'_\ell = \{(\ell - \ell', \ell') : 1 \leq \ell' \leq \min(\ell, |T_{v'}|) \text{ and } 1 \leq \ell - \ell' \leq \min(\ell, |T_{v,i+1}|)\}$$

with the same total cardinality. Their union is a set

$$\begin{aligned} M' = \{(\ell - \ell', \ell') : & 1 \leq \ell \leq \min(k, |T_{v,i}|), \\ & 1 \leq \ell' \leq \min(\ell, |T_{v'}|) \text{ and} \\ & 1 \leq \ell - \ell' \leq \min(\ell, |T_{v,i+1}|)\} \end{aligned}$$

$$\begin{aligned} = \{(\ell - \ell', \ell') : & 1 \leq \ell - \ell' + \ell' \leq \min(k, |T_{v,i}|), \\ & 1 \leq \ell' \leq \min(\ell - \ell' + \ell', |T_{v'}|) \text{ and} \\ & 1 \leq \ell - \ell' \leq \min(\ell - \ell' + \ell', |T_{v,i+1}|)\} \end{aligned}$$

$$\begin{aligned} = \{(x, y) : & 1 \leq x + y \leq \min(k, |T_{v,i}|), \\ & 1 \leq y \leq \min(x + y, |T_{v'}|) \text{ and} \\ & 1 \leq x \leq \min(x + y, |T_{v,i+1}|)\} \end{aligned}$$

$$= \{(x, y) : 1 \leq x + y \leq k, 1 \leq y \leq |T_{v'}| \text{ and } 1 \leq x \leq |T_{v,i+1}|\}$$

To obtain the last equality we made use of the fact that $|T_{v,i}| = |T_{v'}| + |T_{v,i+1}|$. We can now weaken the $1 \leq x + y \leq k$ constraint to $1 \leq x, y \leq k$ to conclude that $|M'| \leq \min(k, |T_{v'}|) \cdot \min(k, |T_{v,i+1}|)$. Similar analysis shows that the same bound also holds for *SAME*, hence completing the proof. □

Before proving the stronger $O(nmk)$ bound, we first show an easier bound of $O(n^2 m)$, which is tight when $k = n$ and better than $O(nmk^2)$ whenever $k = \omega(n^{1/2})$. Proving this is both informative in itself, helping to build intuition, and will also provide us with a tool useful for the general argument. The $O(n^2 m)$ bound will be immediate from the following lemma:

Lemma 4.11. *Let c be such that $1 \leq c \leq m$. Then, solving all subproblems of the form $dp_2[-, -, -, c]$ using Lemma 4.8 takes time $O(n^2)$.*

Proof. By Lemma 4.10, the time required to solve all subproblems of the form $dp_2[-, -, -, c]$ is

asymptotically bounded by

$$\sum_{\substack{1\le v\le n\\ 1\le i\le n_v}} (\min(k,|T_{v'}|)\cdot\min(k,|T_{v,i+1}|)) \le \sum_{\substack{1\le v\le n\\ 1\le i\le n_v}} (|T_{v'}|\cdot|T_{v,i+1}|)$$

where v' denotes $child[v,i]$, as usual.

The quantity $|T_{v'}|\cdot|T_{v,i+1}|$ on the right-hand side is of particular interest. Namely, it has a nice combinatorial interpretation: $|T_{v'}|\cdot|T_{v,i+1}|$ counts the number of pairs of vertices (v_1,v_2) such that $v_1\in T_{v'}$ and $v_2\in T_{v,i+1}$ (e.g. Cygan [11]). Note that, for all these pairs, the LCA of v_1 and v_2 is v and that, if we sum this quantity over all $1\le i\le n_v$, we get precisely the number of unordered pairs of distinct vertices whose LCA is v. It is now immediate that, if we further sum this quantity over all $1\le v\le n$, we get precisely $\binom{n}{2}$, which is the number of unordered pairs of distinct nodes in a tree with n vertices, completing the proof. □

Theorem 4.12. *Solving all subproblems of the form $dp_2[-,-,-,-]$ using Lemma 4.8 takes time $O(n^2m)$. As a result, CCTP can be solved in time $O(n^2m)$.*

Proof. The first part is immediate by summing up $O(n^2)$ from Lemma 4.11 over all $1\le c\le m$. CCTP can be solved in time $O(n^2m)$ because subproblems of this form dominate the complexity, as pointed out earlier. □

We conclude the section by proving the $O(nmk)$ time bound. Once again, it will be an immediate consequence of the following lemma:

Lemma 4.13. *Let c be such that $1\le c\le m$. Then, solving all subproblems of the form $dp_2[-,-,-,c]$ using Lemma 4.8 takes time $O(nk)$.*

Proof. Let us return to the expression for the time S needed to solve all subproblems of the form mentioned:

$$S=\sum_{\substack{1\le v\le n\\ 1\le i\le n_v}} (\min(k,|T_{v'}|)\cdot\min(k,|T_{v,i+1}|)) \tag{4.3.1}$$

Note that the pairs (v,i) over which the sum is taken can be thought of as corresponding to the edges of the tree, one term of the sum corresponding to each edge. With this in mind, we now develop a new way of computing S based on incrementally building the tree starting from n singleton vertices:

1. Start with $S=0$ and an empty graph consisting of n disconnected singleton vertices.
2. At each, step pick an edge (v,v') of the tree which has not been chosen before and connect the corresponding vertices in the graph. Of course, this edge corresponds to a pair (v,i) such that $v'=child[v,i]$, as discussed above. We call such an operation a *join*. We only permit a join to take place if all pairs (v,i') with $i'>i$ have already been picked and the connected component of v' in the graph is isomorphic to $T_{v'}$.
3. Increase S by $\min(k,|T_{v'}|)\cdot\min(k,|T_{v,i+1}|)$.
4. Repeat until all edges of the tree have been added to the graph (i.e. the graph has become isomorphic to the tree).

Observe the following:

- At each stage the graph is a forest of trees, all of which are of the form $T_{v,i}$ for different values of v and i.
- Intuitively, the validity conditions outlined in step 2 above say that valid orders of joining the connected components of the graph until it becomes isomorphic with the tree correspond to valid orders of solving all the subproblems of the form $dp_2[-,-,-,c]$ (see the proof of Theorem 4.9 for a technical definition of such orders). As mentioned before, this does not include the time it takes to solve other subproblems they may depend on (specifically, those with a larger value of c).
- There are multiple ways to carry out this process and, at the end, S will have the same value, given in equation (4.3.1), no matter which one is chosen.

These being said, we can choose the way in which to carry out this construction so as to facilitate bounding the value of S. More precisely, we will split the process into two phases:

1. In this phase we only allow ourselves to pick edges (v, v') for which $|T_{v'}| \leq k$ and $|T_{v,i+1}| \leq k$. In other words, we only allow joining together "small" connected components. We stop only when any further join would involve at least one connected component of size greater than k.
2. We perform the rest of the joins arbitrarily.

We can now write S as $S_1 + S_2$, corresponding to the amounts added to it in the first and second phases, respectively. Our task now is to show that S_1 is $O(nk)$ and that S_2 is $O(nk)$, facts which will be proven using essentially different techniques:

- Proof that S_1 *is* $O(nk)$:
  At the end of phase 1, the graph will consist of a forest of some number p of trees: T_{v_1,i_1}, T_{v_2,i_2}, ..., T_{v_p,i_p}. This state of the graph corresponds to having solved all subproblems of the form $dp_2[v,i,-,c]$ on which $dp_2[v_1,i_1,-,c], \ldots, dp_2[v_p,i_p,-,c]$ depend (possibly indirectly, and including the problems themselves) and no others. This is the same as performing the complete algorithm restricted to each of T_{v_1,i_1}, T_{v_2,i_2}, ..., T_{v_p,i_p} individually. This allows us to bound S_1 asymptotically by applying Lemma 4.11 to each connected component individually and summing up the results:

  $$S_1 \leq |T_{v_1,i_1}|^2 + |T_{v_2,i_2}|^2 + \ldots + \left|T_{v_p,i_p}\right|^2$$

  Since each such connected component has been generated by joining together two connected components of size at most k, we can bound their individual sizes by $2k$:

  $$|T_{v_1,i_1}|^2 + |T_{v_2,i_2}|^2 + \ldots + \left|T_{v_p,i_p}\right|^2 \leq 2k \cdot (|T_{v_1,i_1}| + |T_{v_2,i_2}| + \ldots + \left|T_{v_p,i_p}\right|) = 2nk$$

  And so $S_1 \leq 2nk$, which is $O(nk)$.
- Proof that S_2 *is* $O(nk)$:
  Each join happening in phase 2 has the property that at least one of the involved connected components has size greater than k. For the purposes of this analysis, we will look at the sizes of the connected components of the graph as a sequence a, whose entries are $a_1, a_2, \ldots, a_p$ at

the beginning of phase 2. In the spirit of the potential analysis method, define an auxiliary quantity as follows:

$$X = \min(k, a_1) + \min(k, a_2) + \ldots + \min(k, a_p) \leq n$$

Now, consider a join operation merging together two connected components of sizes a_i and a_j. Without loss of generality, $a_i > k$. This operation removes a_i and a_j from a and adds back $a_i + a_j$. This changes the value of X by removing a $\min(k, a_i) + \min(k, a_j) = k + \min(k, a_j)$ term and adding back a $\min(k, a_i + a_j) = k$ term, making for an overall decrease of X by $\min(k, a_j)$. This same operation increases S_2 by $\min(k, a_i) \cdot \min(k, a_j) = k \cdot \min(k, a_j)$. Therefore, whenever X decreases by ΔX, S_2 increases by $k\Delta X$. Since X can only ever decrease, starts off bounded from above by n and never becomes negative, S_2 will be bounded from above by nk, which is $O(nk)$, completing the proof.

□

Theorem 4.14. *Solving all subproblems of the form $dp_2[-,-,-,-]$ using Lemma 4.8 takes time $O(nmk)$. As a result, CCTP can be solved in time $O(nmk)$.*

Proof. The first part is immediate by summing up $O(nk)$ from Lemma 4.13 over $1 \leq c \leq m$. CCTP can be solved in time $O(nmk)$ because subproblems of this form dominate the complexity. □

5. Towards a Polynomial-Time Algorithm for CC on Grid Single-Crossing Preferences and Median Graphs

We now consider CC for preferences that are single-crossing on arbitrary median graphs. Having already solved the tree case, the next most interesting median graphs are the d-grids. For simplicity, we will only treat the case $d = 2$ here, but note that all presented ideas easily generalize to higher dimensions. We introduce the idea of sliceable tilings and then provide a polynomial-time algorithm under the assumption that optimal solutions correspond to sliceable tilings. Finally, we drop this assumption and show that, in a different sense than usual, our algorithm is still a good approximation.

5.1 A Conjecture and Polynomial-Time Approximation

Assume that $C = [m]$ and that the voters $V = [n_1] \times [n_2]$ define an $n_1 \times n_2$ grid. Additionally, as per Remark 2.22, assume that the r values of each voter are distinct. We will use $r(i, j, c)$ to denote the dissatisfaction of voter (i, j) with candidate c. All our claims apply to both utilitarian and egalitarian CC, but, to keep notation simple, we will only present the former.

We begin by proving a key property of optimal k-assignments w_{opt}:

Lemma 5.1. *For any optimal k-assignment w_{opt} and any candidate c, it holds that the inverse image $w_{opt}^{-1}(c)$ defines a subrectangle of the grid V.*

In other words, w_{opt} induces a partition of the grid into at most k subrectangles.

Proof. Let R be the smallest rectangle that contains voters $w_{opt}^{-1}(c)$ (i.e. their bounding box). Assume for a contradiction that there is a voter $(i, j) \in R$ such that $w_{opt}(i, j) = c' \neq c$. Then, there are 2 voters $v_0, v_1 \in R \cap w_{opt}^{-1}(c)$ for which there is a shortest $v_0 - v_1$ path P passing through (i, j). Since w_{opt} is optimal, it means that $c \succ_{v_0} c'$ and $c \succ_{v_1} c'$, but $c' \succ_{(i,j)} c$, contradicting single-crossingness over P. The fact that such v_0, v_1 do exist is non-trivial and can be shown by contradiction—assuming they did not exist leads to R not being the smallest. □

The importance of this result stands in the fact that any optimal k-assignment w_{opt} can now be more succinctly expressed through a set F_{opt} of at most k subrectangles which partition the grid. All voters in a given rectangle $R \in F_{opt}$ share a common representative candidate c which is implicitly chosen to minimize their total dissatisfaction. Let us call any partition of V into at most

k subrectangles a *k-tiling* (i.e. F_{opt} is a k-tiling). As a result, in the same spirit as our reduction from CC to CCTP (see Chapter 4), we will from now on assume that solving CC means finding a k-tiling F_{opt} that minimizes the total dissatisfaction of the voters in the implicitly associated k-assignment.

Definition 5.2. Let F be a k-tiling of V. We call F *sliceable* if either of the following holds:

1. F consists of a single rectangle.
2. F can be partitioned into two sets of rectangles F_ℓ and F_r such that F_ℓ is a sliceable tiling of $[n_1] \times [1 : j]$ and F_r is a sliceable tiling of $[n_1] \times [j + 1 : n_2]$, for some $j \in [n_2 - 1]$.
3. F can be partitioned into two sets of rectangles F_u and F_d such that F_u is a sliceable tiling of $[1 : i] \times [n_2]$ and F_d is a sliceable tiling of $[i + 1 : n_1] \times [n_2]$, for some $i \in [n_1 - 1]$.

Intuitively, F is sliceable if it can be recursively subdivided with vertical/horizontal lines until we get to singleton rectangles. We now state a conjecture under which a polynomial-time algorithm for CC becomes feasible:

Conjecture 5.3. All optimal k-tilings are sliceable.

Experimental evidence suggests that this is not unreasonable: the conjecture always holds for $n_1 \leq 4$, $n_2 \leq 5$ and $k \leq 5$ (as well as other small instances—see Appendix for a discussion). Additionally, for $k \leq 4$ there is a direct proof that it always holds. Furthermore, it can be shown to always hold under the additional assumption that the preferences of every pair of voters adjacent in the grid differ in at most one pair of candidates.

Assuming that the conjecture holds, we now propose a DP algorithm for grid CC. Define the subproblems to be of the form $dp[i_0, i_1, j_0, j_1, \ell]$, representing the minimal possible dissatisfaction of an ℓ-tiling of voters $[i_0 : i_1] \times [j_0 : j_1]$. They are properly defined for $1 \leq i_0 \leq i_1 \leq n_1$, $1 \leq j_0 \leq j_1 \leq n_2$ and $1 \leq \ell \leq k$.

We now present the base case and recurrence relations that will be used to evaluate the DP:

Lemma 5.4. *Define the following three quantities:*

$$CONST = \min_{1 \leq c \leq m} \sum_{i=i_0}^{i_1} \sum_{j=j_0}^{j_1} r(i, j, c) \tag{5.1.1}$$

$$VERT = \min_{\substack{j_0 \leq j < j_1 \\ 1 \leq \ell' < \ell}} (dp[i_0, i_1, j_0, j, \ell'] + dp[i_0, i_1, j + 1, j_1, \ell - \ell']) \tag{5.1.2}$$

$$HOR = \min_{\substack{i_0 \leq i < i_1 \\ 1 \leq \ell' < \ell}} (dp[i_0, i, j_0, j_1, \ell'] + dp[i + 1, i_1, j_0, j_1, \ell - \ell']) \tag{5.1.3}$$

Then, it holds that $dp[i_0, i_1, j_0, j_1, \ell] = \min(CONST, VERT, HOR)$.

Proof. The proof is very similar to some of our earlier proofs, so we only sketch the main details. Observe that *CONST*, *VERT* and *HOR* correspond to the three clauses of Definition 5.2. The base case *CONST* computes the optimal dissatisfaction attainable by a single rectangle. This is done by iterating through all possible candidates and finding the corresponding dissatisfaction in each case. *VERT* picks a vertical separation line and a number ℓ' and assumes that the tiling is split into ℓ' rectangles to its left and $\ell - \ell'$ rectangles to its right. The situation is analogous for *HOR*. The conjecture guarantees that this is enough to get the right answer. □

Note that the recurrences do not forbid a given candidate from being assigned to more than one rectangle in a tiling, but, like in the previous chapter when we discussed CCTP, this does not lead to a problem since this is never the case for optimal tilings (Lemma 5.1).

Theorem 5.5. *Assuming that Conjecture 5.3 holds, grid single-crossing CC can be solved in time polynomial in n_1, n_2, m, k.*

Proof. We process the subproblems in increasing order of $i_1-i_0+j_1-j_0+\ell$, solving each subproblem by directly applying the recurrence relations in Lemma 5.4. For the time complexity, consider a fixed subproblem and note that computing *VERT* and *HOR* together takes time $O(n_1k + n_2k)$ and that computing *CONST* takes time $O(n_1n_2m)$. Therefore, the total time complexity is $O(n_1^2n_2^2k(n_1k + n_2k + n_1n_2m))$, sufficing to prove the claim. This can be slightly improved by being more diligent about how *CONST* is computed, but we omit these details for now. □

This is not very useful with no clue as to whether the conjecture holds true. However, we can prove some approximation guarantees even without assuming the conjecture. We begin by introducing some definitions and notation:

Definition 5.6. Let F and F' be tilings of V. We say that F' *refines* F if any rectangle in F can be written as the union of some rectangles in F'.

Proposition 5.7. *Let F and F' be tilings of V such that F' refines F, then the total dissatisfaction of F' is at most that of F (i.e. refinement can not increase dissatisfaction).*

Definition 5.8. Let F be a tiling of V and let $\mathcal{L}$ be a vertical/horizontal line between two lines/columns of the grid. Define $split_{\mathcal{L}}(F)$ to be the set F where each rectangle whose interior intersects $\mathcal{L}$ has been replaced by the two "halves" $\mathcal{L}$ splits it into.

Observe that $split_{\mathcal{L}}(F)$ refines F. Also, if F is k-tiling, then $split_{\mathcal{L}}(F)$ is a $2k$-tiling.

Definition 5.9. Given a tiling F, think of its rectangles as actual geometric objects in $\mathbb{Z}^2$. In particular, a given rectangle $[i_0 : i_1]\times[j_0 : j_1] \in F$ has corners at coordinates $(x, y) \in \{(i_0-1, j_0-1)$, $(i_0 - 1, j_1)$, $(i_1, j_0 - 1)$, $(i_1, j_1)\}$.[1] With this in mind, define $P(F)$ to be the set of all corners of all rectangles in F. Likewise, define $x(F) = \{x : (x, y) \in P(F)\}$ and $y(F) = \{y : (x, y) \in P(F)\}$.

Lemma 5.10. *Let F be a k-tiling of V, then there exists a sliceable k^2-tiling F' which refines F.*

Proof. For simplicity, denote $|x(F)|$ by l_x and $|y(F)|$ by l_y. Additionally, assume that $x(F) = \{x_1, \ldots, x_{l_x}\}$ and $y(F) = \{y_1, \ldots, y_{l_y}\}$, where $x_1 < \ldots < x_{l_x}$ and $y_1 < \ldots < y_{l_y}$. Initialize F' to F and perform the following steps:

1. For each vertical line $\mathcal{L}$ defined by an element of $y(F)$, replace F' by $split_{\mathcal{L}}(F')$.
2. For each horizontal line $\mathcal{L}$ defined by an element of $x(F)$, replace F' by $split_{\mathcal{L}}(F')$.

Clearly, F' refines F, since each individual operation is a refinement. Furthermore, F' consists of precisely $(l_x - 1)(l_y - 1)$ rectangles, each with a lower left corner at (x_i, y_j) and an upper right corner at (x_{i+1}, y_{j+1}), for some $1 \le i < l_x$ and $1 \le j < l_y$. Since F is a k-tiling, it follows that $l_x \le k + 1$ and $l_y \le k + 1$, implying that $(l_x - 1)(l_y - 1) \le k^2$, and so F' is a k^2-tiling. It is not hard to see that F' is also sliceable, by definition, hence completing the proof. □

[1] To keep consistent with the way grids are normally drawn, the OX axis should be regarded as going from north to south and the OY axis from east to west.

An approximation guarantee now follows as a direct consequence:

Corollary 5.11. By running our algorithm with k^2 instead of k, we are guaranteed to find a committee that is at least as good as the optimum one for k. Note that this generalizes to k^d for d-grids, becoming less attractive for higher dimensions.

We conclude by noting that an alternative proof of Lemma 5.10 only performs splits when necessary and after each split recurses on the two halves of the partition, making them sliceable independently. While this is likely to introduce fewer additional rectangles in practice, without further analysis it may still produce $O(k^2)$-tilings. Unfortunately, there seems to be no easy way of telling if this may happen on a given input instance.

Going back to our conjecture, trying to prove/disprove it is made especially difficult by the fact that most k-tilings can not actually constitute an optimal solution for any input preferences (e.g. while there are non-sliceable 5-tilings of the 3×3 grid, none of them can ever constitute an optimal 5-tiling, as our implementation in the Appendix shows). At this point further insight seems to be needed into the structure of optimal k-tilings.

6. Conclusions and Future Directions

We investigated the complexity of winner determination for the Chamberlin–Courant rule in the case of generalized single-crossing elections: paths, trees and d-grids. For paths, our results are most attractive if we assume that the individual dissatisfactions are positive integers with values bounded by a polynomial in the input size (often the case in practice; e.g. β-CC). In particular, we gave algorithms whose running times are only a polylogarithmic factor away from the input size for both versions of the rule (i.e. $O^*(nm)$). For trees, we pointed out an error in an earlier attempt at the problem and gave the first correct polynomial-time algorithm. Finally, we began the study of Chamberlin–Courant winner determination for median graphs, by considering the d-grids. While the algorithm we gave for this case is only guaranteed to work if a conjecture we make is true, we showed how it can be used in general to produce committees of no worse than optimal cost, but of a larger size. Even if the assumed conjecture will turn out to not hold in general, understanding why this is the case will be essential for producing an unconditionally correct algorithm or a proof of NP-hardness of the problem. We intend to submit our contributions for consideration at the AAAI-21 Conference.

6.1 Reflection

In this section the author will use the singular pronoun "I" to refer to themselves.

The project was always open ended, with the purpose of investigating interesting algorithmic aspects of single-crossingness. Initially, we considered the concept of k-implementable graphs, introduced by Cohen, Elkind and Lakhani [9] (i.e. graphs whose edges correspond to pairs of candidates whose relative order changes more than once as we sweep from left to right through a given list of preferences), one major open problem being whether 3-implementability is decidable in polynomial time. The concept of tree/median graph single-crossingness was also in sight from the beginning. While reading the account of Clearwater, Puppe and Slinko [8] on median graphs I became familiar with the Chamberlin–Courant rule and their extension to the tree case of the algorithm of Skowron et al. [30] for winner determination. However, the time complexity analysis for this extension was not clear to me. I pointed out the possibly exponential behaviour during a meeting with my supervisor and they subsequently messaged the authors to confirm that, indeed, the presented algorithm may work in exponential time on trees with linearly many leaves. In the meantime, I gave the problem a serious thought and came up with the algorithm presented in Chapter 4, which is, to the best of our knowledge, the first correct algorithm for this problem.

Liking the general appeal of this question, I further read about the proof that utilitarian Monroe's rule is NP-hard in the classical sense (Skowron et al. [30]) and proposed that we study the complexity of more proportional representation problems for the single-crossing domains. My

supervisor proposed that this hardness result may potentially be extended to elections which are single-crossing on trees of bounded diameter. Unfortunately, despite best efforts, this problem remains open.

For the following discussion, introduce *concavity* as the property that the optimal total dissatisfaction for a given instance is concave as a function of k. Whenever this holds true, it is almost always the case that time complexity can be improved. Familiar with how concavity-based techniques like the $O(n \log U)$ algorithm of Bein, Larmore and Park [3] can be applied to speeding up DPs, I proposed that maybe the classical algorithm of Skowron et al. [30] can be improved beyond the more immediate $O(nmk)$ algorithm in Section 3.1. Showing concavity directly proved very difficult, and I only succeeded by phrasing the problem as an instance of LPP and proving the stronger property of Monge-Concavity (from which concavity follows, see Schieber [28]). This stronger result allowed using the state-of-the-art for LPP to get bounds better than what we were initially striving for (i.e. $nm2^{O\left(\sqrt{\log k \log\log n}\right)}$ for the utilitarian case).

With paths and trees solved, it was now a natural extension to consider median graphs, of which the simplest are the d-grids, a question which hasn't been tackled before. For this, my methodology can be summarized as using classical methods combined with algorithmic hypothesis testing (see Appendix for an example of this). This way, I arrived at the algorithm in Chapter 5, whose correctness we could neither prove nor disprove.

I conclude by saying that probably the most challenging and yet rewarding part of this experience was getting a holistic understanding of the field of Computational Social Choice—especially when it came to the older papers mostly focusing on economics and politics. Overall, I became aware of many new perspectives and got to know how doing real research looks like in this field.

6.2 Future Directions

First, the problems of deciding 3-implementability and proving NP-hardness of Monroe's rule for trees of bounded diameter are definitely worth further investigation.

Having proved that path instances of utilitarian Chamberlin–Courant satisfy concavity, it is now natural to ask whether the same also holds for tree instances. Unfortunately, Monge-Concavity and LPP are both one-dimensional notions, so new methods will be needed to answer this question. However, if the answer turns out to be affirmative, then our method from Section 4 extends to $O^*(nm)$ in the spirit of the $O(n \log U)$ algorithm of Bein, Larmore and Park [3].

For the path case a direct proof of Monge-Concavity would be reassuring—one interesting approach is investigating the LP duality certificates of infeasibility in search for a general argument.

Our notion of sliceable tilings resembles what Kusters and Speckmann [18] call *sliceable rectangular duals* and is closely related to that of *fault-free* domino tilings. Results like theirs on characterizing adjacency graphs of rectangle tilings may be instrumental in understanding the fine structure of optimal k-tilings.

More generally, it would be interesting to extend our result for trees to graphs of bounded treewidth and to consider parameterized complexity too. Finally, and probably most importantly, it would be interesting to see which of our ideas extend to single-peaked elections and in what form.

Bibliography

[1] Alok Aggarwal, Baruch Schieber, and Takeshi Tokuyama. "Finding a minimum-weight k -link path in graphs with the concave Monge property and applications". In: *Discrete Computational Geometry* 12 (Dec. 1994), pp. 263–280. DOI: 10.1007/BF02574380.

[2] Kenneth Arrow. *Social Choice and Individual Values*. John Wiley and Sons, 1951.

[3] Wolfgang Bein, Lawrence Larmore, and James Park. "The d-edge shortest-path problem for a Monge graph". In: *UNT Digital Library* (1992). URL: https://digital.library.unt.edu/ark:/67531/metadc1317498/.

[4] Nadja Betzler, Arkadii Slinko, and Johannes Uhlmann. "On the Computation of Fully Proportional Representation". In: *Journal of Artificial Intelligence Research* 47 (Nov. 2011). DOI: 10.2139/ssrn.1952497.

[5] Duncan Black. "On the Rationale of Group Decision-making". In: *Journal of Political Economy* 56.1 (1948), pp. 23–34.

[6] Duncan Black. *The Theory of Committees and Elections*. Springer Netherlands, 1986. ISBN: 978-94-010-8375-1,978-94-009-4225-7.

[7] John Chamberlin and Paul Courant. "Representative Deliberations and Representative Decisions: Proportional Representation and the Borda Rule". In: *American Political Science Review* 77.3 (1983), pp. 718–733. DOI: 10.2307/1957270.

[8] Adam Clearwater, Clemens Puppe, and Arkadii Slinko. "Generalizing the Single-Crossing Property on Lines and Trees to Intermediate Preferences on Median Graphs". In: *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI'15. Buenos Aires, Argentina: AAAI Press, 2015, pp. 32–38. ISBN: 9781577357384.

[9] Nathann Cohen, Edith Elkind, and Foram Lakhani. "Single-crossing Implementation". In: *ArXiv* abs/1906.09671 (June 2019).

[10] Thomas Cormen et al. "The potential method". In: *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. Chap. 17.3, pp. 459–463. ISBN: 0262033844.

[11] Marek Cygan. "Barricades". In: *Looking for a Challenge?* Ed. by Krzysztof Diks et al. 2012, pp. 63–67.

[12] Gabrielle Demange. "Majority relation and median representative ordering". In: *SERIEs: Journal of the Spanish Economic Association* 3.1 (Mar. 2012), pp. 95–109. DOI: 10.1007/s13209-011-0052-9.

[13] Edith Elkind, Martin Lackner, and Dominik Peters. "Structured Preferences". In: *Trends in Computational Social Choice*. Ed. by Ulle Endriss. AI Access, 2017. Chap. 10, pp. 187–207. ISBN: 9781326912093.

[14] Edith Elkind et al. “Properties of Multiwinner Voting Rules”. In: *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*. AAMAS'14. Paris, France: International Foundation for Autonomous Agents and Multiagent Systems, June 2014, pp. 53–60. ISBN: 9781450327381.

[15] Piotr Faliszewski et al. “Multiwinner Voting: A New Challenge for Social Choice Theory”. In: *Trends in Computational Social Choice*. Ed. by Ulle Endriss. AI Access, 2017. Chap. 2, pp. 27–47. ISBN: 9781326912093.

[16] Felix Fischer et al. “Weighted Tournament Solutions”. In: *Handbook of Computational Social Choice*. Ed. by Felix Brandt et al. Cambridge University Press, 2016, pp. 85–102. DOI: 10.1017/CBO9781107446984.005.

[17] Florian Jaeckle, Dominik Peters, and Edith Elkind. “On Recognising Nearly Single-Crossing Preferences”. In: *AAAI*. 2018.

[18] Vincent Kusters and Bettina Speckmann. “Towards Characterizing Graphs with a Sliceable Rectangular Dual”. In: Sept. 2015, pp. 460–471. ISBN: 978-3-319-27260-3. DOI: 10.1007/978-3-319-27261-0_38.

[19] Jean-François Laslier. “And the Loser Is... Plurality Voting”. In: *Electoral Systems: Paradoxes, Assumptions, and Procedures*. Ed. by Dan S. Felsenthal and Moshé Machover. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 327–351. ISBN: 978-3-642-20441-8. DOI: 10.1007/978-3-642-20441-8_13.

[20] Tyler Lu and Craig Boutilier. “Budgeted Social Choice: From Consensus to Personalized Decision Making.” In: *IJCAI International Joint Conference on Artificial Intelligence*. Jan. 2011, pp. 280–286. DOI: 10.5591/978-1-57735-516-8/IJCAI11-057.

[21] Nicholas Mattei and Toby Walsh. “PrefLib: A Library for Preferences http://www.preflib.org”. In: *Algorithmic Decision Theory*. Ed. by Patrice Perny, Marc Pirlot, and Alexis Tsoukiàs. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 259–270. ISBN: 978-3-642-41575-3.

[22] James Mirrlees. “An Exploration in the Theory of Optimum Income Taxation”. In: *The Review of Economic Studies* 38.2 (1971), pp. 175–208.

[23] Burt Monroe. “Fully Proportional Representation”. In: *American Political Science Review* 89.4 (1995), 925â940. DOI: 10.2307/2082518.

[24] Henry Mulder. “The Structure of Median Graphs”. In: *Discrete Mathematics* 24 (Dec. 1978), pp. 197–204. DOI: 10.1016/0012-365X(78)90199-1.

[25] Annette Pilkington. “Topic 2 : Plurality and Runoff Methods for choosing a winner”. In: *Math* 10170 *Mathematics in Sport*. University of Notre Dame, Spring 2016. URL: https://www3.nd.edu/~apilking/math10170/Information/Lectures.htm.

[26] Ariel Procaccia, Jeffrey Rosenschein, and Aviv Zohar. “On the complexity of achieving proportional representation”. In: *Social Choice and Welfare* 30 (Feb. 2008), pp. 353–362. DOI: 10.1007/s00355-007-0235-2.

[27] Kevin Roberts. “Voting over income tax schedules”. In: *Journal of Public Economics* 8.3 (1977), pp. 329–340. ISSN: 0047-2727. DOI: https://doi.org/10.1016/0047-2727(77)90005-6.

[28] Baruch Schieber. “Computing a minimum-weight k-link path in graphs with the concave Monge property”. In: *J. Algorithms* 29 (1995), pp. 204–222.

[29] Piotr Skowron, Piotr Faliszewski, and Arkadii Slinko. "Achieving fully proportional representation: Approximability results". In: *Artificial Intelligence* 222 (2015), pp. 67–103. ISSN: 0004-3702. DOI: 10.1016/j.artint.2015.01.003.

[30] Piotr Skowron et al. "The complexity of fully proportional representation for single-crossing electorates". In: *Theoretical Computer Science* 569 (2015), pp. 43–57. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2014.12.012.

[31] Reza Zanjirani Farahani and Masoud Hekmatfar, eds. *Facility location: Concepts, models, algorithms and case studies*. Jan. 2009.

7. Appendix: Code

This appendix lists two pieces of C++ code:

- Section 7.1 showcases calling the Z3 Theorem Prover on all satisfiability instances described in Section 3.3, hence proving algorithmically that all single-crossing instances of utilitarian CC satisfy Monge-Concavity.
- Section 7.2 showcases a sample of the code written to test various properties of grid single-crossing profiles. As listed here, it attempts to disprove two hypotheses:
  - That Conjecture 5.3 holds, stating that all optimal k-tilings are sliceable. The search confirms that the conjecture holds on a variety of small input instances: $(n_1, n_2, k) \in \{(8,8,4), (4,5,5), (3,6,5), (3,3,6)\}$. The crucial aspect which allowed these checks to be performed is that it is enough to consider the case $m = k$, as explained below.
  - That in every optimal k-tiling each rectangle touches at least one of the sides of the grid. The search disproved this, finding a small preference profile on which it does not hold (see code for the details). The reason this property was considered in the first place is that it allowed for a proof that Conjecture 5.3 holds in general.

7.1 LP Proof of Monge-Concavity for Single-Crossing Utilitarian CC

```
/*
 * Calling the Z3 Theorem Prover on each of the 151 linear programs described in Section 3.3.
 * Requires an installation of the Z3 Theorem Prover and compiling with the -lz3 linker flag.
 */
#include <bits/stdc++.h>
#include <z3++.h>

using namespace std;
using namespace z3;

void check(const int p2[5], const int p3[5], const int c2) {
    // Make variables - r(v, c) will be denoted by get_expr(v, c).
    // A workaround is used to get around the fact that expr is not default
    // constructible (e.g. expressions like r[{2, 3}] would not compile).
    context cx;
    map<pair<int, int>, expr> r;
    auto get_expr = [&r](const int v, const int c) {
        return r.find(make_pair(v, c))->second;
    };
    for (int v = 1; v <= 3; ++v) {
        for (int c = 1; c <= 4; ++c) {
            r.insert(make_pair(make_pair(v, c),
              cx.real_const(("r_" + to_string(v) + "_" + to_string(c)).c_str())));
        }
```

```
    }

    // Add constraints.
    solver s(cx);
    // Premise (5).
    s.add(get_expr(1, 1)     == 0);
    s.add(get_expr(2, p2[1]) == 0);
    s.add(get_expr(3, p3[1]) == 0);
    for (int c = 1; c + 1 <= 4; ++c) {
        s.add(get_expr(1, c)     <= get_expr(1, c + 1));
        s.add(get_expr(2, p2[c]) <= get_expr(2, p2[c + 1]));
        s.add(get_expr(3, p3[c]) <= get_expr(3, p3[c + 1]));
    }
    // Condition (2) in Lemma 3.10.
    for (int c = 1; c <= 4; ++c) {
        for (int c1 = 1; c1 <= 4; ++c1) {
            // Or just ">", equivalently, as per Remark 3.11.
            // Both options give "unsat" verdicts, as expected.
            s.add(get_expr(1, c) + get_expr(2, c) + get_expr(2, c1) + get_expr(3, c1)
              >= 1 + get_expr(1, c2) + get_expr(2, c2) + get_expr(3, c2));
        }
    }
    if (s.check() != unsat) {
        cout << s.get_model() << endl;
        exit(1);
    }
}

// Govern a preference profile P = (id, sigma1, sigma2) returns whether P is single-crossing.
bool single_crossing(const int sigma1[5], const int sigma2[5]) {
    int where1[5], where2[5];
    for (int i = 1; i <= 4; ++i) {
        where1[sigma1[i]] = where2[sigma2[i]] = i;
        }
    for (int a = 1; a <= 4; ++a) {
        for (int b = a + 1; b <= 4; ++b) {
            if (where1[a] > where1[b] && where2[a] < where2[b]) {
                return false;
            }
        }
    }
    return true;
}

int main() {
    // List all triples (sigma1, sigma2, c2) such that P = (id, sigma1, sigma2) is single-crossing.
    int sigma1[5] = {0, 1, 2, 3, 4};
    do {
        int sigma2[5] = {0, 1, 2, 3, 4};
        do {
            if (single_crossing(sigma1, sigma2)) {
                static int cnt = 0; ++cnt;
                cerr << "Processing profile " << cnt << endl;  // 151 in total.
                for (int c2 = 1; c2 <= 4; ++c2) {
                    check(sigma1, sigma2, c2);
                }
            }
        } while (next_permutation(sigma2 + 1, sigma2 + 5));
    } while (next_permutation(sigma1 + 1, sigma1 + 5));
    return 0;
}
```

7.2 Grid Single-Crossing CC Hypothesis Testing

```
/*
 * Testing two hypotheses about the structure of optimal k-tilings of grid single-crossing
 * preferences. The main observation used is that:
```

```
 *   If a certain property fails for an optimal k-tiling on a given problem instance, then
 *   it also fails on the same instance if all candidates not part of the elected
 *   committee are removed. This means that it is enough to consider the case k = C
 *   and only look at the most preferred candidate of each voter.
 *
 *
 * Hypothesis 1: All optimal k-tilings are sliceable.
 * Results:
 *   Confirmed for:
 *     N <= 8, M <= 8 and C = 4
 *     N <= 4, M <= 5 and C = 5
 *     N <= 3, M <= 6 and C = 5
 *     N <= 3, M <= 3 and C = 6
 *   Additionally confirmed for
 *     N <= 6, M <= 6 and C = 6
 *   under the assumption that neighboring preferences differ
 *   by at most one pair of candidates.
 *
 * Hypothesis 2: All rectangles in an optimal k-tiling touch the sides of the grid.
 * Result: Not true for N = M = 3, C = 5 and the following preference profiles profiles:
 *   01234 02134 03214
 *   12304 21304 32104
 *   41230 42130 43210
 *
 *
 * Notation and technical assumptions:
 *   Voters are pairs of integers from the set {0, ..., N - 1} x {0, ..., M - 1}.
 *   Candidates are integers from the set {0, ..., C - 1}.
 *   Individual preferences are denoted by lists of candidates (e.g. {0, 2, 1} means that 0 > 2 > 1).
 *   Grid preference profiles are denoted by lists of lists of individual preferences (e.g. {{p1, p2},
 *   {p3, p4}} means that voter (0, 0) has preferences p1, ..., voter (1, 1) has preferences p4).
 *   Without loss of generality, voter (0, 0) prefers candidates in order 0 > 1 > ... > C - 1.
 */
#include <bits/stdc++.h>

using namespace std;

// Individual preference list.
using Pref = vector<int>;
// Placeholder for unknown preference lists.
const Pref EmptyProf = vector<int>();

// Given a preference list p, returns the index of candidate c (i.e. 0
// if c is first in the list, 1 if c is second in the list, and so on).
int pos(const Pref& p, const int c) {
    for (int i = 0; i < static_cast<int>(p.size()); ++i) {
        if (p[i] == c) {
            return i;
        }
    }
    throw logic_error("Could not find candidate.");
}
// Given a preference list p, returns whether candidate c0 is prefered over candidate c1.
bool prefers(const Pref& p, const int c0, const int c1) {
    return pos(p, c0) < pos(p, c1);
}
// Given two preference lists p0 and p1, returns the number of unordered pairs of candidates
// (c0, c1) such that c0 is prefered to c1 in p0, but c1 is prefered to c0 in p1, or vice-versa.
int cnt_crosses(const Pref& p0, const Pref& p1, const int C) {
    vector<int> inv_p0(C), inv_p1(C);
    for (int i = 0; i < C; ++i) {
        inv_p0[p0[i]] = inv_p1[p1[i]] = i;
    }
    int ans = 0;
    for (int c0 = 0; c0 < C; ++c0) {
        for (int c1 = c0 + 1; c1 < C; ++c1) {
            ans += ((inv_p0[c0] < inv_p0[c1]) != (inv_p1[c0] < inv_p1[c1]));
        }
    }
    return ans;
```

```
}

const int INF = numeric_limits<int>::max();

// Data structure for maintaining bounding boxes. Supports adding points, unioning
// and checking whether the interior intersects a given horizontal/vertical line.
struct Rect {
    int r0, r1;
    int c0, c1;
    Rect(int _r0 = INF, int _r1 = -INF, int _c0 = INF, int _c1 = -INF):
        r0(_r0), r1(_r1), c0(_c0), c1(_c1) {}
    Rect add(const int r, const int c) {
        return Rect(min(r0, r), max(r1, r), min(c0, c), max(c1, c));
    }
    // Returns whether the rectangle intersects the horizontal line between rows r and r + 1.
    bool intersects_with_horizontal(const int r) {
        return r0 <= r && r < r1;
    }
    // Returns whether the rectangle intersects the vertical line between columns c and c + 1.
    bool intersects_with_vertical(const int c) {
        return c0 <= c && c < c1;
    }
};

// Given two bounding boxes r0 and r1, returns whether their intersection is non-empty.
bool do_intersect(const Rect& r0, const Rect& r1) {
    if (r0.r0 > r1.r1) {
        return false;
    } else if (r1.r0 > r0.r1) {
        return false;
    } else if (r0.c0 > r1.c1) {
        return false;
    } else if (r1.c0 > r0.c1) {
        return false;
    } else {
        return true;
    }
}

// Preference profile - a two-dimensional array of preference lists
// (some of which are potentially unknown).
using Grid = vector<vector<Pref>>;

// Prints a preference profile g to stdout.
void show(const Grid& g) {
    const int N = g.size();
    assert(N > 0);
    const int M = g[0].size();
    assert(M > 0);
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            if (g[i][j] == EmptyProf) {
                cout << "?";
            } else {
                for (int k = 0; k < static_cast<int>(g[i][j].size()); ++k) {
                    cout << g[i][j][k];
                }
            }
            cout << " ";
        }
        cout << endl;
    }
    cout << "####" << endl;
}

// Given a preference profile g and two candidates c0 and c1, returns the
// bounding box of all voters which prefer c0 to c1 in g.
Rect get_preference_bounding_box(const Grid& g, const int c0, const int c1) {
    const int N = g.size();
    assert(N > 0);
```

```
    const int M = g[0].size();
    assert(M > 0);
    Rect ans;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            if (g[i][j] != EmptyProf && prefers(g[i][j], c0, c1)) {
                ans = ans.add(i, j);
            }
        }
    }
    return ans;
}

// Given a preference profile g and a candidate c, returns the bounding
// box of all voters for which c is their most preferred candidate.
Rect get_dominance_box(const Grid& g, const int c) {
    const int N = g.size();
    assert(N > 0);
    const int M = g[0].size();
    assert(M > 0);
    Rect ans;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            if (g[i][j][0] == c) {
                ans = ans.add(i, j);
            }
        }
    }
    return ans;
}

// Given a (potentially incomplete) preference profile g, returns false
// if and only if it is certain that it is not single-crossing.
bool grid_valid(const Grid& g, const int C) {
    for (int c0 = 0; c0 < C; ++c0) {
        for (int c1 = c0 + 1; c1 < C; ++c1) {
            if (do_intersect(get_preference_bounding_box(g, c0, c1),
                             get_preference_bounding_box(g, c1, c0))) {
                return false;
            }
        }
    }
    return true;
}

// Given a preference profile g, returns whether all voters have
// the same most preferred candidate.
bool is_monodominated(const Grid& g) {
    const int N = g.size();
    assert(N > 0);
    const int M = g[0].size();
    assert(M > 0);
    // It is enough to check whether all voters' most preferred candidate is 0
    // since we assumed that voter (0, 0) prefers candidates in order 0 > ... > C - 1.
    Rect r = get_dominance_box(g, 0);
    return r.r0 == 0 && r.c0 == 0 &&
           r.r1 == N - 1 && r.c1 == M - 1;
}

// Given a preference profile g, returns whether the dominance box (as computed by a
// call to "get_dominance_box") of some candidate does NOT touch the four sides of the grid.
bool has_isolated(const Grid& g, const int C) {
    const int N = g.size();
    assert(N > 0);
    const int M = g[0].size();
    assert(M > 0);
    for (int c = 0; c < C; ++c) {
        Rect r = get_dominance_box(g, c);
        if (r.r0 == INF) {  // c is not the most preferred candidate of any voter.
            continue;
```

```
        }
        if (r.r0 > 0 && r.r1 < N - 1 && r.c0 > 0 && r.c1 < M - 1) {
            return true;
        }
    }
    return false;
}

// Given a preference profile g, returns whether there exists a horizontal/vertical
// line which does not intersect the dominance box of any candidate. Note that this
// is the same as the tiling formed by these dominance boxes admitting a split line
// (which is the first condition for a non-trivial sliceable tiling).
bool admits_split_line(const Grid& g, const int C) {
    const int N = g.size();
    assert(N > 0);
    const int M = g[0].size();
    assert(M > 0);
    for (int i = 0; i + 1 < N; ++i) {
        bool ok = true;
        for (int c = 0; c < C && ok; ++c) {
            Rect r = get_dominance_box(g, c);
            if (r.intersects_with_horizontal(i)) {
                ok = false;
            }
        }
        if (ok) {
            return true;
        }
    }
    for (int j = 0; j + 1 < M; ++j) {
        bool ok = true;
        for (int c = 0; c < C && ok; ++c) {
            Rect r = get_dominance_box(g, c);
            if (r.intersects_with_vertical(j)) {
                ok = false;
            }
        }
        if (ok) {
            return true;
        }
    }
    return false;
}

// Given a (potentially incomplete) preference profile g, returns whether there are two
// voters adjacent in the grid whose preferences differ in more than one pair of candidates.
bool grid_has_fast_cross(const Grid& g, const int C) {
    const int N = g.size();
    assert(N > 0);
    const int M = g[0].size();
    assert(M > 0);
    for (int i = 0; i + 1 < N; ++i) {
        for (int j = 0; j < M; ++j) {
            if (g[i][j] != EmptyProf && g[i + 1][j] != EmptyProf && cnt_crosses(g[i][j], g[i + 1][j], C) > 1) {
                return true;
            }
        }
    }
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j + 1 < M; ++j) {
            if (g[i][j] != EmptyProf && g[i][j + 1] != EmptyProf && cnt_crosses(g[i][j], g[i][j + 1], C) > 1) {
                return true;
            }
        }
    }
    return false;
}

// Backtracking search - given a (potentially incomplete) grid preference profile g and
// the coordinates of the first voter (r, c) whose preferences have not yet been decided,
```

```
// explores the space of complete grid single-crossing profiles which agree with g.
// For each complete single-crossing profile we test our hypotheses.
void backtr(Grid& g, const int C, const int r, const int c) {
    const int N = g.size();
    assert(N > 0);
    const int M = g[0].size();
    assert(M > 0);

    // Experiment modifier: only test grids for which adjacent voters vary in preference
    // by at most one pair of candidates.
    /*if (grid_has_fast_cross(g, C)) {
        return;
    }*/

    // Prune profiles which can not be single-crossing early.
    if (!grid_valid(g, C)) {
        return;
    } else if (r == N) {
        // Monitor progress.
        static int cnt = 0; ++cnt;
        if (cnt % 100 == 0) {
            cerr << "Processed " << cnt << " grid profiles." << endl;
        }
        // Print grids considered.
        //show(g);

        // Hypothesis 1: All optimal k-tilings are sliceable.
        //   N, M, C = 8, 8, 4 OK.
        //   N, M, C = 4, 5, 5 OK.
        //   N, M, C = 3, 6, 5 OK.
        //   N, M, C = 3, 3, 6 OK.
        //   N, M, C = 6, 6, 6 OK (for no "fast crosses").
        if (!admits_split_line(g, C) && !is_monodominated(g)) {
            show(g);
            exit(1);
        }

        // Hypothesis 2: All rectangles in an optimal k-tiling touch the sides of the grid.
        // Does not hold on the following instance:
        //   01234 02134 03214
        //   12304 21304 32104
        //   41230 42130 43210
        /*if (has_isolated(g, C)) {
            show(g);
            exit(1);
        }*/
    } else if (c == M) {
        backtr(g, C, r + 1, 0);
    } else {
        g[r][c].resize(C);
        iota(g[r][c].begin(), g[r][c].end(), 0);
        do {
            backtr(g, C, r, c + 1);
            // The first voter is assumed to always have preferences 0 > ... > C - 1.
            if (r == 0 && c == 0) {
                break;
            }
        } while (next_permutation(g[r][c].begin(), g[r][c].end()));
        g[r][c] = EmptyProf;
    }
}

int main() {
    const int N = 4;
    const int M = 5;
    const int C = 5;
    Grid g(vector<vector<Pref>>(N, vector<Pref>(M, EmptyProf)));
    backtr(g, C, 0, 0);
    return 0;
}
```