Outer Diversity of Structured Domains

Piotr Faliszewski
AGH University
Kraków, Poland
faliszew@agh.edu.pl

Krzysztof Sornat
AGH University
Kraków, Poland
sornat@agh.edu.pl

Stanisław Szufa
CNRS, LAMSADE, Université Paris Dauphine–PSL
Paris, France
s.szufa@gmail.com

Tomasz Wąs
University of Oxford
Oxford, United Kingdom
tomasz.was@cs.ox.ac.uk

Abstract

An ordinal preference domain is a subset of preference orders that the voters are allowed to cast in an election. We introduce and study the notion of *outer diversity* of a domain and evaluate its value for a number of well-known structured domains, such as the single-peaked, single-crossing, group-separable, and Euclidean ones.

Keywords

Diversity, Ordinal Elections, Structured Domains, Single-Peaked, Single-Crossing.

ACM Reference Format:
Piotr Faliszewski, Krzysztof Sornat, Stanisław Szufa, and Tomasz Wąs. 2026. Outer Diversity of Structured Domains. In *Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), Paphos, Cyprus, May 25 – 29, 2026*, IFAAMAS, 17 pages.

1 Introduction

In the standard, ordinal model of elections, each voter considers a set of candidates and ranks them from the one that he or she likes most to the one that he or she likes least. In principle, a voter may order the candidates in any arbitrary way, but some of these rankings appear more natural (or, more rational) than others. For example, in the political setting it would be expected that a voter would rank the candidates with respect to their proximity to his or her political stance, but a ranking with the most right-wing candidate and the most left-wing one on two top positions would be surprising. Various rationality conditions for ordinal rankings are expressed as so-called *structured domains*, i.e., sets of rankings that can be cast in a given setting. Such domains include, e.g., the single-peaked one [5], which captures preferences based on proximity to some ideal, the single-crossing ones, introduced in the context of taxation [31, 34], or group-separable ones [25, 26], where voters derive rankings of candidates from preferences over their features [18, 27]. We introduce a new measure of diversity of such domains, provide algorithms for computing its value, and analyze diversity of a number of structured domains.

Somewhat surprisingly, analysis of diversity for structured domains has only recently started to receive more focused attention [1, 21, 28], with a few authors also considering diversity of elections [17, 19, 23]. Two commonly used approaches are:

Richness Diversity. The overarching idea is that a domain is diverse if it contains many different substructures in its rankings (these substructures are sometimes also called *attributes*, as the approach builds on the theory of attribute diversity of Nehring and Puppe [32]). For example, one might consider how many votes appear in the domain, how many candidates are ever ranked on top, or—for each triple of candidates—how many ways of ranking these candidates appear in the domain. This approach is taken, e.g., by Ammann and Puppe [1] and Karpov et al. [28]

Inner Diversity. In this case, we say that a domain is diverse if its rankings do not form clear clusters. This approach was taken by Faliszewski et al. [17, 19, 21], who introduced the k-Kemeny problem to quantify the difficulty of clustering rankings (briefly put, one tries to optimally partition the rankings into a given number of groups, measuring their cohesiveness using the classic Kemeny rule [29]).

We propose a third approach, which we refer to as *outer diversity*:

Outer Diversity. A domain is diverse if, on average, a random ranking from the space of all possible ones is similar to some ranking from the domain. In particular, we measure similarity between rankings using the number of swaps of adjacent candidates that transform one into the other.

Inner and outer diversity seem to capture the same basic intuition, but the inner approach focuses on the rankings within the domain, whereas the outer one focuses on those outside.

We believe that all the above approaches to measuring domain diversity are meaningful and are worth studying, but outer diversity has some advantages. First, it has a very clear interpretation: If a domain has high diversity, then it covers the space of all possible rankings well; if one wanted to cast a ranking from the domain but had one that did not belong to it, then the closest member of the domain would not be too far off from his or her original ranking.

Second, outer diversity of a given domain is a single number. On the other hand, in case of richness diversity one has to choose from many different substructures to count, and in case of inner diversity one either has to choose the number of clusters to consider (for which there is no clear solution) or somehow aggregate

Proc. of the 25th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2026), C. Amato, L. Dennis, V. Mascardi, J. Thangarajah (eds.), May 25 – 29, 2026, Paphos, Cyprus.

obtained values for different numbers of clusters, which is not obvious (indeed, the works we cite with respect to inner diversity do not provide fully satisfying recommendations).

Third, while in principle computing outer diversity may require exponential time, we provide efficient algorithms for computing it using sampling: Our algorithms compute the distance from a given ranking to the closest one in a given domain of interest, such as the single-peaked, single-crossing, and group-separable ones. Hence, we can sample random votes, compute their distances to the domains, and output the average of the obtained values. On the other hand, even the heuristics that Faliszewski et al. [17] proposed for inner diversity (i.e., for k-Kemeny) require exponential time if a given domain contains exponentially many rankings (as is the case for, e.g., the single-peaked and group-separable ones).

Our main contributions are as follows:

(1) We introduce the notion of outer diversity and provide means of computing its values for a number of domains, including the single-peaked, single-crossing, and group-separable ones, but also many others (including variants of the single-peaked domain, as well as Euclidean domains). However, we also find that for some natural domains, the sampling-based approach requires solving an NP-hard problem.

(2) We evaluate outer diversity across a number of domains. We find that ranking the domains with respect to outer diversity gives similar results as doing so with respect to the inner one. Further, while analyzing outer diversity of our domains, we note a number of their interesting features.

(3) We compute domains of given sizes, whose outer diversity is (close to) the highest possible, and we analyze how close are various structured domains to these maximal values.

One of the takeaway messages of our work is that the domain of group-separable preferences based on caterpillar trees (see Section 2) is the most diverse one among those that we study, and has many features that other domains often lack. Consequently, and strengthening the message of Faliszewski et al. [21], we believe that this domain should be used in numerical experiments on elections. Even if it does not capture reality in a given setting, it is so special that studying it may lead to the discovery of hard-to-spot phenomena.

We discuss related work throughout the paper, whenever relevant. Omitted proofs are available in the appendix.

2 Preliminaries

For a positive integer t, by $[t]$ we mean the set $\{1, 2, \dots, t\}$. Given an undirected graph G, by $V(G)$ and $E(G)$ we mean its sets of vertices and edges, respectively. We use the *Iverson bracket* notation, i.e., for a logical formula φ, by $[\varphi]$ we mean 1 if φ is true, and 0, otherwise.

Preference Orders, Domains, and Elections. Let C be a set of m *candidates*. By $\mathcal{L}(C)$ we denote the set of all $m!$ linear orders over C, typically referred to as *preference orders*, *votes*, or *rankings*. For each such ranking v and two candidates $a, b \in C$, we write $a \succ_v b$ to indicate that v ranks a ahead of b (i.e., according to v, a is preferred to b). A *preference domain* (over C) is a subset D of $\mathcal{L}(C)$. In particular, $\mathcal{L}(C)$ is the *general domain*. For a ranking v and candidate c, by $\mathrm{pos}_v(c)$ we mean the position of c in v; the top candidate has position 1, the next one has position 2, and so on.

An election is a pair $E = (C, V)$, where $C = \{c_1, \dots, c_m\}$ is a set of candidates and $V = (v_1, \dots, v_n)$ is a collection of voters, each of whom has a vote from $\mathcal{L}(C)$. To streamline the discussion, we use the same symbol v_i to refer both to the given voter and to his or her vote. The exact meaning will always be clear from the context. Given a domain $D \subseteq \mathcal{L}(C)$, we say that $E = (C, V)$ is a D-election if all the voters in V have votes from D.

Sometimes it is convenient to treat a domain $D \subseteq \mathcal{L}(C)$ as an election that contains a single voter for each of its preference orders. In particular, we write UN to mean an election that contains one copy of every possible order (so UN is simply $\mathcal{L}(C)$, viewed as an election). For other domains, we typically do not introduce a second name, but UN has already been used in preceding literature in the context of the map of elections [35].

For two rankings $u, v \in \mathcal{L}(C)$, their *swap distance* (also known as *Kendall's τ distance*) is a number of pairs of candidates in C on whose ordering u and v disagree, i.e.:

$$\mathrm{swap}(u, v) = |\{a, b \in C : a \succ_u b \land b \succ_v a\}|.$$

The value $\mathrm{swap}(u, v)$ can be computed in time $O(m\sqrt{\log m})$ [8]. For a domain $D \subseteq \mathcal{L}(C)$, we let $\mathrm{swap}(D, v) = \min_{u \in D} \mathrm{swap}(u, v)$.

Structured Domains. Let us fix a size-m set of candidates $C = \{c_1, \dots, c_m\}$. Below, we describe the preference domains over C whose diversity we want to analyze.

Consider a connected, undirected graph G, such that $V(G) = C$ (we refer to such graphs as SP-graphs, or SP-trees in case G is also acyclic). A ranking $v \in \mathcal{L}(C)$ is *single-peaked* with respect to G if for every $t \in [m]$, the subgraph induced by the t top-ranked candidates from v is connected. $\mathrm{SP}(G)$ is the domain that consists of all rankings that are single-peaked with respect to G (see, e.g., the work of Elkind et al. [12]). We focus on the following variants:

(1) SP is the classic single-peaked domain that consists of rankings single-peaked with respect to a path (often called an *axis* and denoted $c_1 \rhd c_2 \rhd \cdots \rhd c_m$). In politics, the axis may, e.g., indicate the progression from the most left-wing candidate to the most right-wing one. SP is due to Black [5].

(2) SPOC, introduced by Peters and Lackner [33], consists of rankings single-peaked with respect to a cycle. SPOC preferences appear, e.g., when people located in different time zones want to choose a convenient time for an online meeting. The name SPOC stands for *single-peaked on a circle.*

(3) SP/DF is a domain introduced by Faliszewski et al. [21] and consists of votes single-peaked with respect to a tree that we obtain by taking a path and adding four vertices: two directly connected to one end of the path, and two directly connected to the other end. The name SP/DF stands for *single-peaked/double-forked.* Domains of rankings single-peaked with respect to trees were introduced by Demange [10].

Whenever we speak of SP, SPOC, or SP/DF the exact number of candidates and their positions in respective graphs will be clear from the context (or will be irrelevant). We use this convention for the other domains as well, omitting such details from their names.

A domain is *single-crossing* if it is possible to list its members as $v_1, v_2, \dots, v_n$, so that, as we consider them from v_1 to v_n, the relative ordering of each pair of candidates a and b changes at most once. Single-crossingness is due to Mirrlees [31] and Roberts [34].

(4) By SC, we mean a single-crossing domain sampled from the space of all such domains using the algorithm of Szufa et al. [35]: We generate votes iteratively, starting with some arbitrary vote v_0. In each iteration, given vote v_i, we form v_{i+1} by taking v_i's copy and swapping a randomly selected pair of adjacent candidates that were not swapped in preceding iterations. Altogether, we generate rankings $v_0, \ldots, v_{\binom{m}{2}}$ that form our domain.

Note that the algorithm of Szufa et al. [35] does not sample single-crossing domains uniformly at random (so far, the only known algorithm for such uniform sampling requires exponential time).

Let T be an ordered, rooted tree, where each internal node has at least two children and each leaf is labeled with a unique candidate from C (we refer to such trees as GS-trees). A *frontier* of T is the ranking of the candidates, obtained by reading the leaves of T from left to right. Domain $\mathrm{GS}(T)$ consists exactly of those rankings $v \in \mathcal{L}(C)$ that are either a frontier of T or a frontier of a tree obtained from T by reversing the order of some nodes' children. A domain D is *group-separable* if $D = \mathrm{GS}(T)$ for some T. We are particularly interested in the following two such domains:

(5) GS/bal is a group-separable domain defined by balanced binary trees, i.e., binary trees where each internal node has exactly two children and for each two leaves, their distance from the root differs at most by 1.

(6) GS/cat is a group-separable domain defined by caterpillar binary trees, i.e., trees where each internal node has exactly two children, of which at least one is a leaf.

Group-separable domains were introduced by Inada [25, 26], but the above tree-based definition is due to Karpov [27].

Let d be some positive integer, and let $x\colon C \rightarrow \mathbb{R}^d$ be a function that associates the candidates with distinct points in $\mathbb{R}^d$. A ranking $v \in \mathcal{L}(C)$ is consistent with x if there is a point $x_v \in \mathbb{R}^d$ such that for each two candidates $a, b \in C$ such that $a \succ_v b$ it holds that the Euclidean distance between x_v and $x(a)$ is smaller than that between x_v and $x(b)$. $D(x)$ is the domain that includes exactly the rankings consistent with x. Such domains are called *Euclidean* and were studied, e.g., by Enelow and Hinich [14, 15]. We focus on:

(7) 1D-Int., 2D-Square, and 3D-Cube, where the position of each candidate is sampled uniformly at random from, respectively, $[-1, 1]$, $[-1, 1]^2$, and $[-1, 1]^3$.

It is well-known that 1D-Int. is also a single-crossing domain, and all its votes are single-peaked with respect to the axis obtained by sorting the positions of the candidates.

SP, SC, all group-separable domains, and 1D-Int. are examples of so-called *Condorcet domains*. That is, for every election with odd number of votes from one of these domains, there is a ranking v of the candidates such that if v ranks some candidate a over some other candidate b, then a strict majority of voters prefers a to b.

Distance Between Elections. *Isomorphic swap distance* between two elections (with the same numbers of candidates and the same numbers of voters) is a measure of their structural similarity, introduced by Faliszewski et al. [20]. We extend it to apply to elections with different numbers of voters (in essence, we pretend to duplicate the votes so that the elections appear to be equal-sized).

Definition 2.1. For two elections $E = (C, V)$ and $F = (B, U)$ such that $|C| = |B|$, where $V = (v_1, \ldots, v_n)$ and $U = (u_1, \ldots, u_k)$, their isomorphic swap distance is defined as follows (the indices of the votes from V are taken modulo n, and the indices of the votes from U are taken modulo k):

$$d_{\mathrm{swap}}(E, F) = \frac{1}{nk} \min_{\pi\colon [nk] \rightarrow [nk]} \min_{\sigma\colon C \rightarrow B} \sum_{i \in [nk]} \mathrm{swap}(\sigma(v_i), u_{\pi(i)}),$$

where π and σ are bijections, and by $\sigma(v_i)$ we mean vote v_i where each candidate $c \in C$ is replaced with candidate $\sigma(c) \in B$.

***k*-Kemeny and Inner Diversity.** Let $E = (C, V)$ be an election and let $R = \{r_1, \ldots, r_k\}$ be a set of preference orders from $\mathcal{L}(C)$. By the Kemeny score of R with respect to election E, we mean:

$$\mathrm{kem}_E(R) = \textstyle\sum_{v \in V} \mathrm{swap}(R, v).$$

In other words, it is the sum of the swap distances of the election's votes to their closest rankings from R. The k-Kemeny score of an election E, denoted k-$\mathrm{kem}(E)$, is the smallest Kemeny score of a size-up-to-k set of rankings for this election. By Kemeny score we mean the 1-Kemeny score. Computing the Kemeny score of a given election is well-known to be hard [3, 24], even for the case of four voters [4, 11]. The notion of the Kemeny score was the original idea of Kemeny [29], whereas the extension to collections of rankings was put forward by Faliszewski et al. [17], in the context of election diversity. Specifically, they claimed that the appropriately normalized weighted sum of an election's k-Kemeny scores (for varying k) captures its diversity. Indeed, the larger an election's k-Kemeny score, the more difficult it is to cluster its votes into k groups, meaning that its votes are quite different from one another. Consequently, these votes are diverse. The same view was taken by Faliszewski et al. [19] and was recently applied to measure the diversity of preference domains by Faliszewski et al. [21]. Specifically, given domain D over size-m candidate set, they defined its Kemeny vector to be:

$$\mathrm{kem}(D) = (1\text{-}\mathrm{kem}(D)/|D|, 2\text{-}\mathrm{kem}(D)/|D|, \ldots, m\text{-}\mathrm{kem}(D)/|D|)$$

and they said that a given domain D_1 is more diverse than another domain D_2 (both over equal-sized candidate sets) if $\mathrm{kem}(D_1)$ dominates $\mathrm{kem}(D_2)$ or is close to dominating it; they did not formalize this notion as they considered only a few domains.

We broadly refer to measures of diversity based on the difficulty of clustering as capturing *inner diversity.*

3 Outer Diversity

Let C be a set of candidates and let $D \subseteq \mathcal{L}(C)$ be a domain over C. By the *average normalized swap distance* of D, denoted $\mathrm{ansd}(D)$, we mean the expected swap distance between a vote chosen from $\mathcal{L}(C)$ uniformly at random and the closest vote in D, divided by the maximal possible distance between two votes in $\mathcal{L}(C)$. Formally:

$$\mathrm{ansd}(D) = \tfrac{1}{m!} \textstyle\sum_{u \in \mathcal{L}(C)} \mathrm{swap}(D, u)/\binom{m}{2}.$$

The largest possible value of $\mathrm{ansd}(D)$ is 0.5, obtained when D consists of a single vote, and the smallest one is 0, obtained for the general domain. To ensure that *outer diversity* of a domain D is

Table 1: For each domain we give its size and the complexity of finding its closest member (in terms of swap distance) to a given input ranking. Running times marked with * do not include the time needed for preprocessing.

Domain	Size	Complexity of Finding Closest Ranking from D
GS(T)	$\leq 2^{m-1}$	$O(m^2)$
GS/cat	2^{m-1}	$O(m \log m)$
GS/bal	2^{m-1}	$O(m \log m)$
SP	2^{m-1}	$O(m^2)$
SP/DF	$2^{m+1} - 16$	$O(m^4)$
SPOC	$m2^{m-2}$	$O(m^2)$
SP(T)	—	$O(km^k)$ k = number of T's leaves
SP(G)	—	NP-com.
SC	$1 + m(m-1)/2$	$O(m^2)^*$
1D-Int.	$1 + m(m-1)/2$	$O(m^2)^*$
2D-Square	$O(m^4)$	$O(m^4)^*$
3D-Cube	$O(m^6)$	$O(m^6)^*$

between 0 and 1 (where 0 means complete lack of diversity and 1 means full diversity), we define it as the following linear transformation of ansd(D).

Definition 3.1. For a domain $D \subseteq \mathcal{L}(C)$, its *outer diversity* is defined as out-div(D) $= 1 - 2 \cdot$ ansd(D).

While outer- and inner diversity notions are based on different principles, they are interrelated in several ways. For example, inner diversity, as defined by Faliszewski et al. [17, 19, 21], relies on analyzing k-Kemeny scores of given elections or domains, whereas ansd(D) is simply the normalized k-Kemeny score of the input domain D, with respect to the UN election. Considered from a different perspective, ansd(D) is equal to the smallest possible isomorphic swap distance between UN and a D-election.

Proposition 3.2. *For every domain $D \subseteq \mathcal{L}(C)$, it holds that* ansd($D$) $= \min_{E \textit{ is a D-election}} d_{\text{swap}}(\text{UN}, E)/\binom{m}{2}$.

Since Faliszewski et al. [17] have shown that proximity to UN is highly correlated with their form of inner diversity, we conclude that both approaches are capturing the same high-level idea.

4 Computing Outer Diversity

For domains over sufficiently small candidate sets, it is possible to compute outer diversity exactly. In the most basic approach, given a domain D over candidate set C, we could simply compute the swap distance between every vote in D and every vote in $\mathcal{L}(C)$. Naturally, this is very inefficient and computing outer diversity of, say, SP with m candidates would require time $O(m! \cdot 2^{m-1} \cdot m\sqrt{\log m})$; the general domain has $m!$ rankings, SP has 2^{m-1} of them, and it takes $O(m\sqrt{\log m})$ time to compute the swap distance [8]. Fortunately, there is a faster approach that given a domain D, for each i forms a set D_i of rankings at swap distance i from D.

Proposition 4.1. *There is an algorithm that given domain D over m candidates (represented by listing its members), computes* out-div(D) *in time $O(m^2 \cdot m!)$.*

To compute outer diversity for larger candidate sets, we resort to sampling. Namely, given a domain D over a size-m candidate set C, we fix a number N, sample N rankings from $\mathcal{L}(C)$, for each sampled ranking v we compute swap(D, v) and output the average of these values, divided by $\binom{m}{2}$. This gives an estimate for ansd(D), based on which we obtain out-div(D). However, to implement this idea efficiently, we need fast algorithms for the following problem: Given a ranking v and a domain D, compute swap(D, v). We dedicate the rest of this section to seeking algorithms for this problem for various domains, and to establishing its complexity.

On the outset, the problem can be even NP-hard. For example, for each set of $4m$ candidates $C = \{c_{i,j} : i \in [4], j \in [m]\}$, let the *4-alignment* domain contain each vote of the form $\{c_{1,1}, \dots, c_{1,m}\} \succ \{c_{2,1}, \dots, c_{2,m}\} \succ \{c_{3,1}, \dots, c_{3,m}\} \succ \{c_{4,1}, \dots, c_{4,m}\}$, in which the order of the candidates, based on their second indices, is identical in each block. Then we have the following hardness result (in essence, for this domain the problem of finding a closest vote in the domain becomes the problem of computing Kemeny score for 4 voters, known to be NP-hard [4, 11]).

Theorem 4.2. *Let D be the 4-alignment domain. Given vote v and integer $d \in \mathbb{N}$ it is NP-complete to decide whether* swap(D, v) $\leq d$.

Despite this negative result, for most of our domains we find efficient algorithms for computing the distance to a given vote (see Table 1). In the following, we always use $C = \{c_1, \dots, c_m\}$ to denote the set of m candidates in the domain under consideration.

4.1 Single-Peaked Domains

Let us first consider the family of single-peaked domains. We note that Faliszewski et al. [16, Theorem 4.5.] already gave a polynomial-time algorithm for computing the distance between SP and a given ranking, but their approach—based on dynamic programming—required $O(m^3)$ time. We improve this algorithm to run in $O(m^2)$ time. The main idea is to use dynamic programming to iteratively compute the distance between a given ranking v and votes that rank more and more bottom candidates as required by SP.

Assume that we are given a vote v and a societal axis $c_1 \rhd c_2 \rhd \cdots \rhd c_m$. For each $\ell, r \in \{0, 1, 2, \dots, m\}$ such that $\ell + r \leq m$, let $C_{\ell,r}$ denote the set of the first ℓ and the last r candidates according to $\rhd$. Formally, we have $C_{\ell,r} = \{c_1, \dots, c_\ell\} \cup \{c_{m+1-r}, \dots, c_m\}$; by convention, for $\ell = 0$ we have $\{c_1, \dots, c_\ell\} = \emptyset$, and for $r = 0$ we have $\{c_{m+1-r}, \dots, c_m\} = \emptyset$. Then, by $U_{\ell,r}$ we denote the set of all votes $u \in \mathcal{L}(C)$ in which (a) candidates from $C_{\ell,r}$ are in the bottom $\ell + r$ positions, and (b) for each $t \in \{m, m-1, \dots, m-\ell-r+1\}$, the top t candidates of u form an interval within $\rhd$. Observe that $U_{0,0} = \mathcal{L}(C)$, whereas if $\ell + r = m$, then $U_{\ell,r} =$ SP. We write $A_{\ell,r}$ to denote the minimal swap distance between v and $u \in U_{\ell,r}$. As we will show, all values of $A_{\ell,r}$ can be computed efficiently in Algorithm 1, using a recursive formula.

Theorem 4.3. *Algorithm 1 computes the distance between a given vote and a single-peaked domain in time $O(m^2)$.*

Proof. For the running time, observe that each of our loops is over at most m elements, and we have at most two levels of nested

Algorithm 1 Distance between a ranking and SP

```
Input: Ranking $v \in \mathcal{L}(C)$, societal axis $c_1 \rhd \cdots \rhd c_m$
    Phase 1, Precomputation:
 1: for $i \in [m]$ do
 2:     $L_{i,i} \leftarrow 0, R_{i,i} \leftarrow 0$
 3:     for $j \in \{i+1, \dots, m\}$ do $L_{i,j} \leftarrow L_{i,j-1} + [c_i \succ_v c_j]$
 4:     for $j \in \{i-1, \dots, 1\}$ do $R_{j,i} \leftarrow R_{j+1,i} + [c_i \succ_v c_j]$
    Phase 2, Main Computation:
 5: $A_{0,0} \leftarrow 0$
 6: for $\ell \in [m-1]$ do $A_{\ell,0} \leftarrow A_{\ell-1,0} + L_{\ell,m}$
 7: for $r \in [m-1]$ do
 8:     $A_{0,r} \leftarrow A_{0,r-1} + R_{1,m+1-r}$
 9:     for $\ell \in [m-r-1]$ do
10:         $A_{\ell,r} \leftarrow \min(A_{\ell-1,r} + L_{\ell,m-r},\ A_{\ell,r-1} + R_{\ell+1,m+1-r})$
11: return $\min_{\ell \in [m]} A_{\ell-1,m-\ell}$
```

loops. Each individual iteration can be completed in time $O(1)$. The final minimum in line 11 requires $O(m)$ time.

Let us now analyze the correctness of the algorithm. For each $i, j \in [m]$, with $i \le j$, we let $L_{i,j}$ be the number of candidates in $\{c_i, c_{i+1}, \dots, c_j\}$ that v ranks below c_i. Consequently, we have that $L_{i,i} = 0$ and, if $i < j$, then either $L_{i,j} = L_{i,j-1} + 1$ (if v ranks c_i ahead of c_j) or $L_{i,j} = L_{i,j-1}$ (otherwise). Similarly, for $j \le i$, $R_{j,i}$ is the number of candidates in $\{c_j, c_{j+1}, \dots, c_i\}$ that v ranks below c_i ($R_{j,i}$ satisfies analogous relations as $L_{i,j}$). The algorithm computes the values of $L_{i,j}$ and $R_{j,i}$ in Phase 1.

Then, in Phase 2, the algorithm computes all the values $A_{\ell,r}$ for $\ell, r \in [m]$ such that $\ell + r \le m - 1$. Let us fix such ℓ and r. We note that every ranking in $U_{\ell,r}$ either ranks c_ℓ or c_{m+1-r} on position $m+1-\ell-r$ (i.e., on the $\ell+r$'th position from the bottom). Indeed, for all rankings in $U_{\ell,r}$ we have that the first $m+1-\ell-r$ candidates form an interval within $\rhd$. However, by definition, all of these candidates, except for the one ranked on position $m-\ell-r+1$, belong to $C \setminus C_{\ell,r}$. Consequently, to form the interval, the candidate on position $m-\ell-r+1$ must be either c_ℓ or c_{m+1-r}. Let $U_{\underline{\ell},r}$ be a subset of votes from $U_{\ell-1,r}$ that additionally have c_ℓ in the position $m+1-\ell-r$. Similarly, let $U_{\ell,\underline{r}}$, be a subset of votes from $U_{\ell,r-1}$ with c_{m+1-r} in the position $m+1-\ell-r$. By the preceding argument, we have that $U_{\ell,r} = U_{\underline{\ell},r} \cup U_{\ell,\underline{r}}$ (if $\ell = 0$, we assume $U_{\underline{\ell},r} = \emptyset$, if $r = 0$, $U_{\ell,\underline{r}} = \emptyset$). Thus, $A_{\ell,r} = \min(\mathrm{swap}(U_{\underline{\ell},r}, v), \mathrm{swap}(U_{\ell,\underline{r}}, v))$.

Let u be a vote in $U_{\underline{\ell},r}$ that minimizes $\mathrm{swap}(u, v)$. Observe that $m-\ell-r$ first candidates in u appear in the same relative order as they appear in v (otherwise, ordering them as in v would decrease the distance). Let u' be a vote obtained from u by ensuring that it ranks its first $m-\ell-r+1$ candidates in the same relative order as in v (in other words, u' is the same as u, except that it might rank c_ℓ some positions earlier). It must be that $u' \in U_{\ell-1,r}$. Moreover, we can show that u' minimizes swap distance to v among rankings in $U_{\ell-1,r}$, i.e., $\mathrm{swap}(u', v) = A_{\ell-1,r}$. Indeed, the first $m-\ell-r+1$ candidates are in the optimal order (the same as in v), and if rearranging the last $\ell+r-1$ candidates could decrease the distance, we could also rearrange them in the same way in u. Now, when we look at the inversions counted in $\mathrm{swap}(v, u)$, we see that we count all inversions that we count in $\mathrm{swap}(v, u')$ and additionally those from having c_ℓ after all of the first $m-\ell-r$ candidates. But those are exactly the inversions we store in $L_{\ell,m-r}$. Thus, we get that $\mathrm{swap}(U_{\underline{\ell},r}, v) = A_{\ell-1,r} + L_{\ell,m-r}$. Analogously, we can prove that $\mathrm{swap}(U_{\ell,\underline{r}}, v) = A_{\ell,r-1} + R_{\ell+1,m+1-r}$. This way, we obtain the recursive equation used in line 10, as well as the equations from lines 6 and 8 (in their cases either $r = 0$ or $\ell = 0$ so respective parts of the equation disappear).

Finally, for $\ell \in [m]$, we observe that $A_{\ell-1,m-\ell}$ is the minimal distance from v to a single-peaked ranking u in which c_ℓ is the top candidate. Thus, to get the overall smallest distance, we take the minimum from all these values. □

Every vote in SPOC is single-peaked along the axis obtained by "cutting" the cycle between some two adjacent candidates [33]. There are m such axes, hence we can run Algorithm 1 for each of them and choose the minimum distance. This gives as an algorithm running in time $O(m^3)$. We can improve that and get an $O(m^2)$ algorithm by a similar dynamic programming algorithm as for SP.

Theorem 4.4. *There is an algorithm that computes the swap distance between a given vote and* SPOC *in time* $O(m^2)$.

We can also extend Algorithm 1 to work for the case of $\mathrm{SP}(T)$, where T is an SP-tree. If T has k leaves (i.e., k nodes of degree 1), then the algorithm requires $O(km^k)$ time. The main idea is to implement dynamic programming over sets of connected vertices in T, of which there are $O(m^k)$.

Theorem 4.5. *There is an algorithm that given an* SP*-tree that has k leaves, computes the swap distance between a given vote and* $\mathrm{SP}(T)$ *in time* $O(km^k)$.

Given the algorithms for SP, SPOC, and single-peaked-on-a-tree domains, one could ask for a general polynomial-time algorithm that works for all single-peaked-on-a-graph domains. We prove that in this general case the problem is NP-complete.

Theorem 4.6. *Given a graph G, a vote* $v \in \mathcal{L}(V(G))$*, and an integer* $d \in \mathbb{N}$*, deciding if* $\mathrm{swap}(SP(G), v) \le d$ *is* NP*-complete.*

4.2 Group-Separable Domains

For a group-separable domain with an arbitrary tree, we show an algorithm that computes the distance to a given vote in time $O(m^2)$.

Assume we are given a vote v and a group separable domain $D = \mathrm{GS}(T)$. Then, observe that finding vote $u \in D$ that minimizes $\mathrm{swap}(u, v)$ is equivalent to reversing the order of some of the children of each internal node of T so that the frontier u of T minimizes $\mathrm{swap}(u, v)$. Moreover, the change in distance we get by reversing the order of the children of one particular node is independent of the configuration of the other nodes. Hence, we can consider internal nodes of tree T one by one, and for each decide in which of the two ways its children should be ordered. Fix such an arbitrary node with k children, and let $C_1, C_2, \dots, C_k$ denote the sets of candidates associated with leaves that are descendants of each of the children, when looking from left to right. This configuration would incur the distance of:

$$\textstyle\sum_{1 \le i < j \le k} |\{(a, b) \in C_i \times C_j : b \succ_v a\}|,$$

while reversing the order gives the distance of:

$$\textstyle\sum_{1 \le i < j \le k} |\{(a, b) \in C_i \times C_j : a \succ_v b\}|.$$

Thus, we compute the values of both sums and choose the configuration that leads to the lower one (or make an arbitrary choice in case of a tie). When considering all internal nodes of T in this way, we check each pair of candidates exactly once. Hence, the running time of this algorithm is $O(m^2)$.

Theorem 4.7. *There is an algorithm that given a* GS*-tree T and a vote v, computes* $\mathrm{swap}(\mathrm{GS}(T), v)$ *in time $O(m^2)$.*

For GS/bal and GS/cat, we give algorithms running in time $O(m \log m)$. Both algorithms follow the general approach outlined above, but for GS/bal we speed up computing inversions using an approach similar to that from the classic Merge Sort algorithm, and for GS/cat we use a special data structure.

Theorem 4.8. *There are algorithms that compute the swap distance between a given vote and* GS/cat *and* GS/bal *(represented via* GS*-trees) in time $O(m \log m)$.*

4.3 Single-Crossing and Euclidean Domains

Both single-crossing and Euclidean domains contain polynomially many votes, so a brute-force algorithm that given a ranking v computes its swap distance to all the rankings in the domain runs in polynomial time. For example, for SC, which contains $O(m^2)$ rankings, it would run in time $O(m^3\sqrt{\log m})$ [8]. However, as we typically want to compute the distance from many votes to our domains, we get better running times via appropriate preprocessing. Briefly put, for each domain $D \in \{\mathrm{SC}, \text{1D-Int.}, \text{2D-Square}, \text{3D-Cube}\}$ we can arrange the rankings from these domains on a tree $T(D)$—or even on a path, in case of 1D-Int. and SC—so that two neighboring rankings are at swap distance one. Then, to compute a distance from a given ranking v to each member of the domain, we compute the distance between v and an arbitrary ranking in the domain, and then traverse the tree, updating the distance on the fly, so for each member of the domain we get its swap distance to v. Building $T(D)$ adds, at most, factor $O(m^2)$ to the complexity of computing the rankings from the domain.

Theorem 4.9. *For each D that is either* SC *or a Euclidean domain, there is an algorithm that given a ranking v and tree $T(D)$ computes* $\mathrm{swap}(D, v)$ *in time $O(|D|)$.*

5 Analysis of the Domains

Let us now analyze the outer diversity of our domains. We first consider the case of 8 candidates, and then we analyze how the outer diversities of our domains change as the number of candidates grows. The case of 8 candidates is interesting for the following, somewhat interrelated, reasons: (1) Faliszewski et al. [21] largely focused on this case, and we want our results to be comparable to theirs; (2) The case of 8 candidates is among the most popular ones in experiments within computational social choice [7]; (3) Considering only 8 candidates allows us to perform exact computations.

Table 2: Size, average normalized swap distance, outer diversity, and size of direct neighborhood (also normalized) of various domains, for the case of 8 candidates. The standard deviation of outer diversity for domains that we need to sample (SC, 1D-Int., 2D-Square, 3D-Cube) is no larger than 0.005 (for ten samples).

Domain D	$\lvert D\rvert$	ansd(D)	out-div(D)	dist-1	dist-1/$\lvert D\rvert$
Vote+Its Rev.	2	0.384	0.232	14	7
GS/cat	128	0.194	0.613	704	5.5
GS/bal	128	0.257	0.486	384	3
SP	128	0.284	0.432	384	3
SP/DF	496	0.239	0.522	968	1.952
SPOC	512	0.196	0.608	1280	2.5
SC	29	0.316	0.368	130.3	4.493
1D-Int.	29	0.311	0.378	134.8	4.648
2D-Square	351	0.217	0.566	988.0	2.815
3D-Cube	2311	0.138	0.724	3878.2	1.678
Largest Cond.	224	0.282	0.435	544	2.429

5.1 Outer-Diversity for Eight Candidates

For each of our domains, in Table 2 we provide its size, average normalized swap distance, outer diversity value, the number of votes in $\mathcal{L}(C)$ that are exactly at swap distance 1 from this domain (we refer to this as the *size of the direct neighborhood*), and the latter number normalized by the size of the domain (we analyze these values later on). Additionally, the table also includes LC domain, i.e., the largest Condorcet domain over 8 candidates, recently discovered by Leedham-Green et al. [30]. Sorting our domains with respect to their outer diversity values gives the following ranking:

$$\underset{0.719}{\text{3D-Cube}} \succ \{\underset{0.613}{\text{GS/cat}}, \underset{0.608}{\text{SPOC}}\} \succ \underset{0.565}{\text{2D-Square}} \succ \underset{0.522}{\text{SP/DF}} \succ \underset{0.486}{\text{GS/bal}} \succ \{\underset{0.435}{\text{LC}}, \underset{0.432}{\text{SP}}\} \succ \{\underset{0.386}{\text{1D-Int.}}, \underset{0.37}{\text{SC}}\}.$$

It is quite interesting that even though LC is the largest Condorcet domain over 8 candidates, its outer diversity is very similar to that of SP, which contains nearly half of the votes, and it is notably lower than outer diversities of GS/cat and GS/bal (both of the same cardinality as SP). However, a closer analysis of this domain confirms that it is not as diverse as one might expect given its size. For example, there are only 4 candidates that are ever ranked first in its votes, and 4 different candidate that are ever ranked last (indeed, the domain has further restrictions along these lines, which we omit due to limited space). Next, we note that our ranking is very similar to an analogous one obtained by Faliszewski et al. [21] based on inner diversity (also for the case of 8 candidates; note in their case there are no specific values measuring diversity and the ranking was obtained by comparing Kemeny vectors of the domains):

$$\text{GS/cat} \succ \text{3D-Cube} \succ \{\text{2D-Square, SPOC}\} \succ \{\text{SP/DF, GS/bal}\} \succ \text{SP} \succ \{\text{SC, 1D-Int.}\}.$$

Both rankings put 3D-Cube and GS/cat as the most diverse domains, and they both put 1D-Int. and SC as the least diverse ones. Further, they both rank domains from the same families identically: SPOC is more diverse than SP/DF, which is more diverse than SP, and GS/cat is more diverse than GS/bal (not to mention the ranking of the Euclidean domains). The fact that 3D-Cube has higher outer diversity than GS/cat, as well as the tie between GS/cat and SPOC, are artifacts of considering only 8 candidates and for larger numbers of candidates these relations change (see Section 5.2).

Below, we analyze two features of our domains that are not directly related to capturing diversity, but which manifest themselves during outer diversity computations and which shed some light on how our domains are arranged within the general domain.

5.1.1 Direct Neighborhoods The size of the direct neighborhood of a domain, normalized by the sizes of this domains, is interesting as it gives some intuition on how the domain is "spread" over $\mathcal{L}(C)$. For example, the domain that consists of a single ranking and its reverse is "maximally spread:" Its two members are as far apart as possible and, as we consider 8 candidates, there are exactly 7 rankings next to each of the domain members, neither of which belongs to the domain. Among our structured domains, GS/cat is the most spread one, with the value of 5.5, and 3D-Cube is the least spread, with the value of 1.678. Hence, members of 3D-Cube are packed quite closely within $\mathcal{L}(C)$. While one could think that this is a consequence of 3D-Cube's large size, $\mathcal{L}(C)$ contains more than 16 rankings for every ranking in 3D-Cube. It is interesting that for some domains the normalized sizes of their direct neighborhoods are appealing, round numbers (such as 3 for GS/bal or 5.5 for GS/cat). For GS/bal and GS/cat, we show that this is not a mere coincidence; for the other domains we leave this issue open.

Proposition 5.1. *Let D be the* GS/bal *domain for $m = 2^k$ candidates. For every ranking $v \in D$ there are exactly $2^{k-1} - 1$ unique ones from $\mathcal{L}(C) \setminus D$ at swap distance 1 from v.*

Proposition 5.2. *Consider* GS/cat *over $m \geq 4$ candidates. For every ranking $v \in$* GS/cat *there are exactly $m - 3$ unique ones from $\mathcal{L}(C) \setminus D$ at swap distance 1 from v, and one ranking from $\mathcal{L}(C)$ that is at swap distance 1 from v and one other ranking in* GS/cat.

5.1.2 Popularity Given a domain $D \subseteq \mathcal{L}(C)$ and a ranking $v \in D$, we define its *popularity*, denoted $\mathrm{pop}(v)$, as the number of rankings from $\mathcal{L}(C)$ for which v is the closest member of D (if for a given ranking $u \in \mathcal{L}(C)$ there are p members of D that are closest to u, then u contributes $1/p$ to the popularity of each of them). The average popularity of a ranking in $|D|$ is equal to $|\mathcal{L}(C)|/|D|$ and by *normalized popularity* of a ranking v we mean the ratio between its popularity and this value. Namely, we have $\mathrm{npop}(v) = \frac{\mathrm{pop}(v)}{|L(C)|/|D|}$. Popularity gives hints on both the internal symmetry of a domain, and the arrangement of its rankings in $\mathcal{L}(C)$. Indeed, the more uniform are the popularity values of the rankings, the more likely it is that they are symmetrically spread within $\mathcal{L}(C)$. On the other hand, a mixture of high and low popularity values suggests that the more popular rankings are on the "outskirts" of the domain, and the less popular ones belong to its "interior." We show the normalized popularities of the rankings in our domains in Figure 1, on the microscope plots of Faliszewski et al. [17].

Remark 5.1. *Let D be a domain. A microscope plot of D presents each ranking from the domain as a dot, whose Euclidean distance from the other dots is as similar to the swap distance between the respective rankings as possible (exact correspondence between Euclidean distances and swap distances is, typically, impossible to achieve, but microscopes still give useful intuitions).*

The plots show some remarkable features of our domains. The first observation is that for both GS/bal and GS/cat, all rankings have equal popularity, equal to the expected one. Indeed, this is a general feature of group separable domains.

Proposition 5.3. *Let $D = \mathrm{GS}(T)$ be a group separable domain over candidate set C. Then, for each $v \in \mathrm{GS}(T)$, $\mathrm{npop}(v) = 1$.*

The other domains show a high variance in popularity among their members. For example, the most popular rankings in SP are the societal axis and its reverse, whereas most rankings in between these two have low popularity. Overall, group-separable domains are perfectly symmetric and clearly stand out.

5.2 Outer Diversity for Larger Candidate Sets

When considering more than eight candidates, we compute outer diversity using the sampling approach, with sample size $N = 1000$ (see Section 4). For each domain, we repeat this computation 10 times, to also obtain standard deviation (it is so small as to be nearly invisible on our plots, which justifies the use of sampling).

In Figure 3, we show how the outer diversity of our domains evolves as a function of the number m of candidates, for $m \in \{2, 3, \dots, 20\}$. In particular, we note that the outer diversity of polynomially-sized Euclidean domains drops much more rapidly than that of the other, exponential-sized, ones. It is also notable how SPOC becomes less diverse than GS/cat (for 9 candidates or more) and how GS/cat becomes the most diverse among our domains (for 12 candidates or more). Further, GS/cat is consistently more diverse than GS/bal. As these two domains are extreme among the group-separable ones (one uses the tallest binary GS-tree and the other one the shortest), we ask if GS/cat is the most diverse group-separable domain and GS/bal is the least diverse one.

It is interesting if outer diversity of our domains eventually approaches zero, or if it stays bounded away from it. As shown below, the former happens, e.g., if the size of the domain is bounded by a constant, whereas the latter happens, e.g., for GS/cat. Hence, outer diversity of a domain may be bounded away from zero even if its size grows notably more slowly than that of the general domain (as a function of the number of candidates).

Proposition 5.4. *Let us fix value k and let $D_2, D_3, \dots$ be a sequence of domains, where each D_m contains at most k rankings over m candidates. Then $\lim_{m \to \infty} \text{out-div}(D_m) = 0$.*

Proposition 5.5. *If the number of candidates is even, then* $\text{out-div}(\mathrm{GS/cat}) > 1/2$.

Proof. Take a GS/cat domain for candidate set $C = \{c_1, \dots, c_m\}$, where m is even, defined via binary caterpillar tree where the leaf closest to the root is c_1, the next one is c_2, and so on.

Let v be some arbitrary ranking from $\mathcal{L}(C)$. To transform it into a member of GS/cat we can, for example, sort its top half in the increasing order of the candidate indices, and sort the bottom half in the decreasing order of candidate indices. As shown by Boehmer et al. [6], ensuring that candidate indices first increase and then decrease is a necessary and sufficient condition for a ranking to belong to GS/cat. The number of swaps needed to implement such sorting in the top half of the ranking is equal to the number of inversions there. Since the expected number of inversions in a random permutation is $\frac{1}{4}n(n-1)$, when considering all votes from $\mathcal{L}(C)$, on average we need to perform $\frac{1}{4}(m/2)(m/2-1) = \frac{1}{16}m(m-$

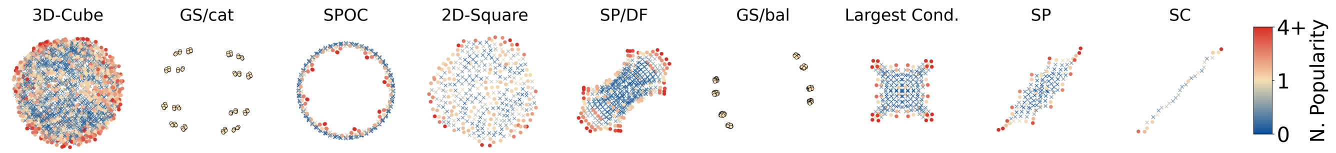

Figure 1: Microscope plots of our domains, where each dot/cross represents a ranking from the domain, colored according to its normalized popularity (see Remark 5.1). Rankings with normalized popularity below 1 are marked with crosses, and the remaining ones with dots. Dots marking rankings with normalized popularity equal to exactly 1 have a black border.

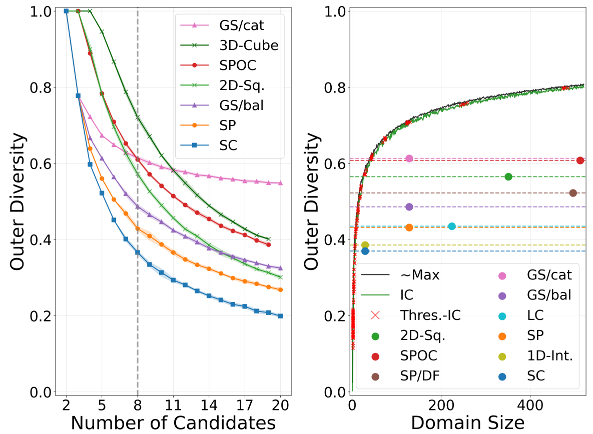

Figure 2: Outer diversity of several structured domains as a function of the number of candidates (on the left), or as a function of their size (on the right; including approximations of most diverse domains). For SPOC and 3D-Cube, we omit outer diversity for 20 candidates, due to computation time.

2) swaps in their top halves, and the same number of swaps in their bottom halves. Altogether, we need to perform $\frac{1}{8}m(m-2)$ swaps per ranking in $\mathcal{L}(C)$, so we have $\mathrm{ansd}(\mathrm{GS/cat}) \leq \frac{m(m-2)/8}{m(m-1)/2} = \frac{1}{4} \cdot \frac{m-2}{m-1}$. This means that we have $\text{out-div}(\mathrm{GS/cat}) \geq 1 - \frac{1}{2} \cdot \frac{m-2}{m-1} > \frac{1}{2}$. □

6 Most Diverse Domains

Given a number k, we ask for a domain of k rankings with the highest outer diversity value. As per our observation in Section 3, we can compute such a domain by solving the k-Kemeny problem for the UN election using, e.g., integer linear programming (ILP).[1] Unfortunately, solving this ILP is challenging, as its size for m candidates is $\Theta((m!)^2)$. Hence, for $m \geq 6$ we use the following heuristics (to compute the outer diversity of the domains produced by them, we use the sampling approach, with $N = 1000$ samples):

(1) We sample k rankings uniformly at random from $\mathcal{L}(C)$ (this is known as sampling from impartial culture, IC).
(2) We sample k rankings from IC and perform simulated annealing (technical details available in the Appendix D.1).

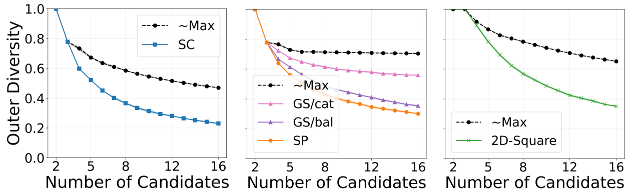

Figure 3: Outer diversity of several structured domains as a function of the number of candidates, compared to the outer diversity of (an approximation of) the most diverse domain of the same size.

We also use a heuristic that does not allow us to control the size of the domain, but selects rankings that are spread out over $\mathcal{L}(C)$:

(3) We choose a threshold $t \in \{5, 6, \ldots, 25\}$ and keep on sampling rankings from IC (altogether 10^4 of them), keeping only those whose swap distance from the closest already-kept one is greater or equal to t.

Instead of using this heuristic, we would rather keep on selecting rankings that are at the largest possible swap distance from those previously selected, but finding such rankings is NP-complete.

Theorem 6.1. *Given a positive integer t and a domain $D \subseteq \mathcal{L}(C)$, represented by explicitly listing its rankings, deciding if there is a ranking v such that $\min_{u \in D} \mathrm{swap}(u, v) \geq t$ is* NP*-complete.*

On the plots, we denote domains computed using the first heuristic as IC, those computed using simulated annealing as ~Max, and those using the threshold approach as Thres.-IC. In Figure 2 (right) we show how the outer diversities of these domains for the case of $m = 8$ candidates, as we increase k (for the first two heuristics) or decrease t (for the third one). We see that for each given size of the domain, all three heuristics produce very similar results. We interpret this as suggesting that, indeed, we get close to the highest possible diversities. For the case of 6 candidates we also compared our heuristically computed domains to the optimal ones, obtained using ILP, and the results were nearly identical (see Appendix D.1). Figure 2 (right) also includes points corresponding to our structured domains, illustrating how far off they are from the most diverse domains of their size.

In Figure 3, for each domain $D \in$ {SC, GS/cat, GS/bal, SP, 2D-Square}, we plot the outer diversity of this domain and the outer diversity of the most diverse domain of size $|D|$ (as computed using our second heuristic) as a function of the number of candidates (for up to 16 of them, as beyond this number computations

[1] Finding k rankings that achieve the optimal k-Kemeny score for UN can be formulated as the k-Median clustering applied on the metric space of all possible rankings under the swap distance. We use the standard ILP formulation for this problem.

proved too intensive). In particular, we see that for polynomial-sized domains (SC and 2D-Square), the diversity of the most diverse domains seems to be dropping up to 16 candidates. In contrast, for SP, GS/bal, and GS/cat, which are all of size 2^{m-1}, the outer diversity of the most diverse domain seems to stabilize around the value 0.7 (indeed, by Proposition 5.5, we know that it cannot go below 0.5; proving a stronger bound would be interesting).

7 Conclusions

Our main conclusion is that outer diversity is a useful, practical measure of domain diversity. Using it, we have found that GS/cat sharply stands out from many other structured domains in various respects and, so, we recommend its use in experiments. Throughout the paper, we have made a number of observations, and we have explained some of them theoretically. We propose seeking such explanations for the remaining observations as future work.

Acknowledgments

Tomasz Wąs was supported by the UK Engineering and Physical Sciences Research Council (EPSRC) under grant EP/X038548/1. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 101002854).

erc
European Research Council
Established by the European Commission

References

[1] M. Ammann and C. Puppe. 2025. Preference Diversity. *Review of Economic Design* (2025). Online First.

[2] C. Baharav, A. Constantinescu, and R. Wattenhofer. 2025. Condorcet Winners and Anscombe's Paradox Under Weighted Binary Voting. In *Proceedings of AAMAS-2025*. 179–187.

[3] J. Bartholdi, III, C. Tovey, and M. Trick. 1989. Voting Schemes for Which it Can Be Difficult to Tell Who Won The Election. *Social Choice and Welfare* 6, 2 (1989), 157–165.

[4] T. Biedl, F. J. Brandenburg, and X. Deng. 2009. On the Complexity of Crossings in Permutations. *Discrete Mathematics* 309, 7 (2009), 1813–1823.

[5] D. Black. 1958. *The Theory of Committees and Elections*. Cambridge University Press.

[6] N. Boehmer, R. Bredereck, E. Elkind, P. Faliszewski, and S. Szufa. 2022. Expected Frequency Matrices of Elections: Computation, Geometry, and Preference Learning. In *Proceedings of NeurIPS-2022*.

[7] N. Boehmer, P. Faliszewski, L. Janeczko, A. Kaczmarczyk, G. Lisowski, G. Pierczyński, S. Rey, D. Stolicki, S. Szufa, and T. Wąs. 2024. Guide to Numerical Experiments on Elections in Computational Social Choice. In *Proceedings of IJCAI-2024*. 7962–7970.

[8] T. Chan and M. Pătraşcu. 2010. Counting Inversions, Offline Orthogonal Range Counting, and Related Problems. In *Proceedings of SODA-10*. 161–173.

[9] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. 2001. *Introduction to Algorithms* (second ed.). MIT Press/McGraw Hill.

[10] G. Demange. 1982. Single-Peaked Orders on a Tree. *Mathematical Social Sciences* 3, 4 (1982), 389–396.

[11] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. 2001. Rank Aggregation Methods for the Web. In *Proceedings of WWW-01*. 613–622.

[12] E. Elkind, M. Lackner, and D. Peters. 2017. Structured Preferences. In *Trends in Computational Social Choice*, U. Endriss (Ed.). AI Access Foundation, 187–207.

[13] E. Elkind, M. Lackner, and D. Peters. 2022. *Preference Restrictions in Computational Social Choice: A Survey*. Technical Report arXiv.2205.09092 [cs.GT]. arXiv.org.

[14] J. Enelow and M. Hinich. 1984. *The Spatial Theory of Voting: An Introduction*. Cambridge University Press.

[15] J. Enelow and M. Hinich. 1990. *Advances in the Spatial Theory of Voting*. Cambridge University Press.

[16] P. Faliszewski, E. Hemaspaandra, and L. Hemaspaandra. 2014. The Complexity of Manipulative Attacks in Nearly Single-Peaked Electorates. *Artificial Intelligence* 207 (2014), 69–99.

[17] P. Faliszewski, A. Kaczmarczyk, K. Sornat, S. Szufa, and T. Wąs. 2023. Diversity, Agreement, and Polarization in Elections. In *Proceedings of IJCAI-2023*. 2684–2692.

[18] P. Faliszewski, A. Karpov, and S. Obraztsova. 2022. The complexity of election problems with group-separable preferences. *Autonomous Agents and Multi-Agent Systems* 36, 1 (2022), 18.

[19] P. Faliszewski, J. Mertlová, P. Nunn, S. Szufa, and T. Wąs. 2025. Distances Between Top-Truncated Elections of Different Sizes. In *Proceedings of AAAI-2025*. 13823–13830.

[20] P. Faliszewski, P. Skowron, A. Slinko, K. Sornat, S. Szufa, and N. Talmon. 2025. How Similar Are Two Elections? *J. Comput. System Sci.* 150 (2025), 103632.

[21] P. Faliszewski, K. Sornat, S. Szufa, and T. Wąs. 2025. *Diversity of Structured Domains via k-Kemeny Scores*. Technical Report arXiv:2509.15812 [cs.GT]. arXiv.org.

[22] M. Garey and D. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

[23] V. Hashemi and U. Endriss. 2014. Measuring Diversity of Preferences in a Group. In *Proceedings of ECAI-2014*. 423–428.

[24] E. Hemaspaandra, H. Spakowski, and J. Vogel. 2005. The Complexity of Kemeny Elections. *Theoretical Computer Science* 349, 3 (2005), 382–391.

[25] K. Inada. 1964. A Note on the Simple Majority Decision Rule. *Econometrica* 32, 32 (1964), 525–531.

[26] K. Inada. 1969. The Simple Majority Decision Rule. *Econometrica* 37, 3 (1969), 490–506.

[27] A. Karpov. 2019. On the number of group-separable preference profiles. *Group Decision and Negotiation* 28, 3 (2019), 501–517.

[28] A. Karpov, K. Markström, S. Riis, and B. Zhou. 2024. *Local Diversity of Condorcet Domains*. Technical Report arXiv:2401.11912 [econ.TH]. arXiv.org.

[29] J. Kemeny. 1959. Mathematics Without Numbers. *Daedalus* 88 (1959), 577–591.

[30] C. Leedham-Green, K. Markström, and S. Riis. 2024. The Largest Condorcet Domain on 8 Alternatives. *Social Choice and Welfare* 62, 1 (2024), 109–116.

[31] J. Mirrlees. 1971. An Exploration in the Theory of Optimal Income Taxation. *Review of Economic Studies* 38 (1971), 175–208.

[32] K. Nehring and C. Puppe. 2002. A Theory of Diversity. *Econometrica* 70, 3 (2002), 1155–1198.

[33] D. Peters and M. Lackner. 2020. Preferences Single-Peaked on a Circle. *Journal of Artificial Intelligence Research* 68 (2020), 463–502.

[34] K. Roberts. 1977. Voting Over Income Tax Schedules. *Journal of Public Economics* 8, 3 (1977), 329–340.

[35] S. Szufa, N. Boehmer, R. Bredereck, P. Faliszewski, R. Niedermeier, P. Skowron, A. Slinko, and N. Talmon. 2025. Drawing a map of elections. *Artificial Intelligence* 343 (2025), 104332.

Algorithm 2 Computing outer diversity by BFS

```
Input: Domain $D$ over candidate set $C = \{c_1, \dots, c_m\}$
1: $D_0 \leftarrow D$, $i \leftarrow 0$
2: while $\bigcup_{j=0}^{i} D_i \neq \mathcal{L}(C)$ do
3:     $D_{i+1} = \emptyset$
4:     for $v \in D_i$ do
5:         for $u$ such that $\mathrm{swap}(u, v) = 1$ do
6:             if $u \notin \bigcup_{j=0}^{i} D_j$: $D_{i+1} \leftarrow D_i \cup \{u\}$
7:     $i \leftarrow i + 1$
8: return $1 - 2(\frac{1}{m!} \sum_{j=0}^{i} j \cdot |D_j|)$
```

A Missing Proof for Section 3

Proposition 3.2. *For every domain $D \subseteq \mathcal{L}(C)$, it holds that* $\mathrm{ansd}(D) = \min_{E \textit{ is a D-election}} d_{\mathrm{swap}}(\mathrm{UN}, E)/\binom{m}{2}$.

Proof. Fix election $E = (C, V)$ yielding the minimal distance. Without loss of generality, we can assume that the number of votes in V is a multiple of $m!$, i.e., $V = \{v_1, \dots, v_{k\cdot m!}\}$ for some $k \in \mathbb{N}$, because creating k additional copies of all votes does not affect the isomorphic swap distance. Let $u_1, \dots, u_{k\cdot m!}$ be copies of voters in UN as denoted in Definition 2.1 and $\pi : [k \cdot m!] \to [k \cdot m!]$ be a matching of voters yielding the minimum distance. We can assume that the matching of candidates, σ, is the identity since for the distance to UN every matching of candidates gives the same sum of distances.

Observe that $\mathrm{swap}(u_i, v_{\pi(i)}) = \mathrm{swap}(D, u_i)$, for every $i \in [k\cdot m!]$ as otherwise $d_{\mathrm{swap}}(\mathrm{UN}, E)$ could be decreased by exchanging $v_{\pi(i)}$ for v yielding the minimum and keeping all other voters as is. This also implies that for each $i \in [m!]$ and $\ell \in [k-1]$, we have $\mathrm{swap}(u_i, v_{\pi(i)}) = \mathrm{swap}(u_{i+\ell\cdot m!}, v_{\pi(i+\ell\cdot m!)})$. Then, we get

$$\begin{aligned}\textstyle\sum_{r \in \mathcal{L}(C)} \mathrm{swap}(D, r) &= \textstyle\sum_{i \in [m!]} \mathrm{swap}(D, u_i)\\ &= \textstyle\sum_{i \in [m!]} \mathrm{swap}(u_i, v_{\pi(i)})\\ &= \tfrac{1}{k} \textstyle\sum_{i \in [k\cdot m!]} \mathrm{swap}(u_i, v_{\pi(i)})\\ &= m! \cdot d_{\mathrm{swap}}(E, \mathrm{UN}),\end{aligned}$$

which yields the thesis. □

B Additional Material for Section 4

In this appendix, we provide further details on algorithmic techniques for establishing outer diversity of given domains.

Our exact algorithm, given as Algorithm 2 proceeds as follows: Given domain D, we form a sequence of sets, $D_0, D_1, \dots$, such that for each i, D_i contains rankings that are at swap distance i from D. For each i, we compute D_{i+1} by considering all the votes that can be obtained from those in D_i by a single swap of adjacent candidates, and include in D_{i+1} those that do not belong to $\bigcup_{j=0}^{i} D_i$. Given D_0, $D_1, \dots$, we compute $\mathrm{ansd}(D)$ as the weighted sum of their sizes, and $\mathrm{out\text{-}div}(D)$ as $1 - 2\mathrm{ansd}(D)$. Fast implementation requires storing each D_i individually, as well as the growing union of these sets, for increasing values of i.

Proposition 4.1. *There is an algorithm that given domain D over m candidates (represented by listing its members), computes* $\mathrm{out\text{-}div}(D)$ *in time $O(m^2 \cdot m!)$.*

Proof. We use Algorithm 2, whose correctness follows directly from the definitions of $\mathrm{ansd}(D)$ and $\mathrm{out\text{-}div}(D)$. In line 6 of the algorithm, for each vote $v \in \mathcal{L}(C)$ we consider all $m-1$ votes u obtained from v by a single swap of adjacent candidates, resulting in $O(m \cdot m!)$ memberships checks. Rankings from $\bigcup_{j=0}^{i} D_j$ are stored in a trie (prefix tree), which allows $O(m)$-time membership checks and insertions. For the current iteration i, we store only sets D_i and D_{i+1} (each taking $O(m \cdot m!)$ space) while retaining the values $|D_j|$ for $j < i$. Hence, the computational bottleneck is line 6 executed $O(m \cdot m!)$ times, each taking $O(m)$ time, leading to a total running time of $O(m^2 \cdot m!)$. □

Theorem 4.2. *Let D be the 4-alignment domain. Given vote v and integer $d \in \mathbb{N}$ it is NP-complete to decide whether* $\mathrm{swap}(D, v) \leq d$.

Proof. The verification is straightforward. Given the vote in D that yields the closest distance to v, we can check whether this distance is larger than d in polynomial time.

To show hardness, we give a reduction from KemenyOn4Votes. In this problem we are given a candidate set $C = \{c_1, \dots, c_m\}$, four votes $v_1, v_2, v_3, v_4 \in \mathcal{L}(C)$, and an integer $d \in \mathbb{N}$, and we ask whether there exists a ranking $u \in \mathcal{L}(C)$ for which it holds that $\sum_{i \in [4]} \mathrm{swap}(v_i, u) \leq d$. This is known to be NP-complete [4, 11].

Now, for each instance of KemenyOn4Votes, let us construct an instance of our problem. To this end, let $C' = \{c_{i,j} : i \in [4], j \in [m]\}$ and let $v \in \mathcal{L}(C')$ be a concatenation of votes v_1, v_2, v_3, and v_4, i.e., $c_{i,j} \succ_v c_{i',j'}$, if and only if, $i < i'$ or $i = i'$ and $c_j \succ_{v_i} c_{j'}$. Also, for every ranking $u \in \mathcal{L}(C)$ let $f(u)$ denote a ranking in $\mathcal{L}(C')$ that is a concatenation of 4 copies of u, i.e., $c_{i,j} \succ_{f(u)} c_{i',j'}$, if and only if, $i < i'$ or $i = i'$ and $c_j \succ_u c_{j'}$. Then, 4-agreement domain can be alternatively written as $D = \{f(u) : u \in \mathcal{L}(C)\}$. Moreover, $\mathrm{swap}(v, f(u)) = \sum_{i \in [4]} \mathrm{swap}(v_i, u)$, for each $u \in \mathcal{L}(C)$. Therefore, indeed, there exists $u \in \mathcal{L}(C)$ such that $\sum_{i \in [4]} \mathrm{swap}(v_i, u) \leq d$, if and only if, $\mathrm{swap}(D, v) \leq d$. □

Next, following the sampling approach, we give detailed descriptions for computing a distance between a given vote v and domain D, where D is either SPOC, SP(T), GS/cat, GS/bal, SC, or Euclidean domain.

B.1 Algorithms for Single-Peaked Domains

Assume we are given a vote v and single-peaked-on-a-cycle domain with cycle $(c_1, \dots, c_m)$. For convenience, we will sometimes allow candidate indices to go over m and treat them as if they cycle over, i.e., $c_{i+m} = c_i$ for each $i \in [m]$.

For each $i \in [m]$ and $j \in \{0, 1, \dots, m-1\}$, let $C_{i,i+j}$ denote the set of candidates $\{c_i, c_{i+1}, \dots, c_{i+j}\}$ that form an interval in the cycle. Then, let $U_{i,i+j}$ be a subset containing all votes u that rank candidates from $C_{i,i+j}$ as top $j+1$ candidates and for each $t \in [j+1]$ the first t candidates in u form an interval in the cycle. Also, let $A_{i,i+j}$ denote the minimum swap distance from v to a vote in $U_{i,i+j}$. These can be efficiently computed using Algorithm 3.

Theorem B.1. *Algorithm 3 computes the distance between a given vote and a single-peaked-on-a-circle domain in time $O(m^2)$.*

Proof. For the running time, we observe that the loops in Algorithm 3 are at most 2-nested, over at most m elements, and each individual iteration runs in time $O(1)$. The final minimum in line 14 runs in time $O(m)$, but it is not part of any loop.

Algorithm 3 Distance between a ranking and SPOC

```
Input: Vote $v \in \mathcal{L}(C)$, societal axis $c_1 \rhd \dots \rhd c_m$
    PHASE 1, PRECOMPUTATION:
 1: for $i \in [m]$ do $c_{i+m} \leftarrow c_i$
 2: for $i \in [m]$ do
 3:     $L_{i,i} \leftarrow 0, R_{i+m,i} \leftarrow 0$
 4:     for $j \in [m-1]$ do $L_{i,i+j} \leftarrow L_{i,i+j-1} + [c_{i+j} \succ_v c_i]$
 5:     for $j \in [m-i-1]$ do $L_{i+m,i+m+j} \leftarrow L_{i,i+j}$
 6:     for $j \in [m-1]$ do $R_{i+m-j,i} \leftarrow R_{i+m-j+1,i} + [c_{i+m-j} \succ_v c_i]$
    PHASE 2, MAIN COMPUTATION:
 7: for $i \in [m]$ do $A_{i,i} \leftarrow L_{i,i+m-1}, \quad A_{m+i,m+i} \leftarrow A_{i,i}$
 8: for $r \in [m-2]$ do
 9:     for $i \in [2m-r]$ do
10:         $A_{i,i+r} \leftarrow \min(A_{i,i+r-1} + L_{i+r,i+m-1},\ A_{i+1,i+r} + R_{i+r+1,i})$
11: return $\min_{i=[m]} A_{i,i+m-2}$
```

For the correctness, similarly as in the proof of Theorem 4.3, we first note that for each $i \in [m], r \in \{0, \dots, m-1\}$ in $L_{i,i+j}$ we store the number of candidates in $\{c_i, c_{i+1}, \dots, c_{i+j}\}$ that are preferred over c_i in v (in Algorithm 1 we counted the number of candidates that are less preferred than c_i, here it is reversed). Analogously, in $R_{i+m-j,i}$, we store the number of candidates preferred over c_i from $\{c_{i+m-j}, c_{i+m-j+1}, \dots, c_{i+m}\}$ (note that it is also an interval). We can efficiently compute both sets of numbers recurrently.

For every $i \in [m]$, $U_{i,i}$ is just the set of all votes that have c_i as the top candidate. Thus, $A_{i,i}$ is just a number of candidates that are preferred over c_i in v, which is what is stored in $L_{i,i+m-1}$ (see line 7 of Algorithm 3).

For $i \in [m]$ and $r \in [m-1]$, we compute values of $A_{i,i+r}$ using a recursive formula in line 10, in a similar way to how it was done in Algorithm 1. Let $U_{\underline{i},i+r}$ be a subset of votes in $U_{i+1,i+r}$ that additionally have c_i at the position $r+1$. Similarly, let $U_{i,\underline{i+r}}$ be a subset of votes in $U_{i,i+r-1}$ with c_{i+r} at the position $r+1$. Since every vote in $U_{i,i+r}$ has candidates $\{c_i, c_{i+1}, \dots, c_{i+r}\}$ at the first $r+1$ positions and the first r candidates form an interval in the cycle, it must be c_i or c_{i+r} in the position $r+1$. Thus, $U_{i,i+r} = U_{\underline{i},i+r} \cup U_{i,\underline{i+r}}$. Hence, $A_{i,i+r} = \min(\min_{u \in U_{\underline{i},i+r}} \mathrm{swap}(v,u), \min_{u \in U_{i,\underline{i+r}}} \mathrm{swap}(v,u))$.

Fix a vote $u \in U_{\underline{i},i+r}$ minimizing $\mathrm{swap}(v,u)$. Observe that the last $m-r-1$ candidates in u have to appear in exactly the same order as they appear in v. Let u' be obtained from u by arranging $m-r$ last candidates in this way (so we additionally relocate c_i). Observe that u' minimizes the swap distance to v among votes in $U_{i+1,i+r}$, i.e., $A_{i+1,i+r} = \mathrm{swap}(u',v)$. Moreover, the difference between $\mathrm{swap}(u,v)$ and $\mathrm{swap}(u',v)$ is the number of candidates outside of $\{c_{i+1}, \dots, c_{i+r}\}$ that are preferred over c_i. Observe that $C \setminus \{c_{i+1}, \dots, c_{i+r}\} = \{c_{i+r+1}, \dots, c_{i+m}\}$. Therefore, we get that $\mathrm{swap}(u,v) = A_{i+1,i+r} + R_{i+r+1,i}$. Analogously, we can prove that $\min_{u \in U_{i,\underline{i+r}}} \mathrm{swap}(v,u) = A_{i,i+r-1} + L_{i+r,i+m-1}$. This yields the recursive equation from line 10.

Finally, observe that for each $i \in [m]$, the set $U_{i,i+m-2}$ contains all single-peaked-on-a-circle votes that have candidate c_{i+m-1} at the bottom of the ranking. Thus, taking the minimum of distances to each such vote we get the minimum distance to any single-peaked-on-a-circle vote. □

We can also extend Algorithm 1 to work on arbitrary tree with k leaves in time $O(km^k)$. The pseudocode is summarized in Algorithm 4.

THEOREM B.2. *Algorithm 4 computes the distance between a given vote and a single-peaked-on-a-tree domain in time $O(km^k)$.*

PROOF. Let us fix such tree G on a set of candidates C and a given vote v. Let $\mathcal{S} = (S_1, S_2, \dots, S_\ell)$ be a sequence of subsets of C such that each $S \in \mathcal{S}$, if and only if, S is nonempty and connected in G, and for each $i, j \in [\ell]$, we have that $S_i \supsetneq S_j$ only if $i < j$. In particular, this means that $S_1 = C$. Observe that the length of the sequence $\mathcal{S}$ is bounded by $O(m^k)$, as each connected subset of C can be uniquely identified by how far away from each leaf is the closest node from S (and the maximal value of such distance is bounded by m). We can also compute such sequence $\mathcal{S}$ in time $O(m^k)$ by checking each possible k-tuple of such distances starting from the smallest ones.

Then, for each $S \in \mathcal{S}$ by $X_S \subseteq S$ let us denote the set of nodes in S that are leafs in the subgraph induced by S (note that there are at most k of them). Furthermore, for each such $x \in X_S$ we denote the number of candidates in S over which x is preferred in v by

$$I_{x,S} = |\{c \in S : c \succ_v x\}|.$$

This corresponds to values $L_{i,j}$ and $R_{i,j}$ used in Algorithm 1. We can compute all of them in time $O(km^k)$ in the reversed order to that in sequence $\mathcal{S}$. This is since for each other leaf $y \in X_S \setminus \{x\}$, it holds that $I_{x,S} = I_{x,S\setminus\{y\}} + [x \succ_v y]$.

Next, for each $S \in \mathcal{S}$ we define U_S as a set of all votes $u \in \mathcal{L}(C)$ in which candidates $C \setminus S$ are in the last positions and for each $t \in [m] \setminus [|S|]$, the first t candidates in u form a connected subset in G. Also, we denote $A_S = \min_{u \in U_S} \mathrm{swap}(u,v)$.

Clearly, $A_{S_1} = 0$ as $S_1 = C$, thus $U_C = \mathcal{L}(C)$. For each $S \in (S_2, \dots, S_\ell)$, we compute A_S recursively, similarly to how we computed $A_{l,r}$ in Algorithm 1. Let $Y_S \subseteq S$ be a subset of nodes in $C \setminus S$ that are connected to some node in S (again, there are at most k of them). Then, for each $y \in Y_S$, we can denote $U_{S,y}$ as a subset of votes in $U_{S\cup\{y\}}$ that have y in the position $|S|+1$. Since every vote in U_S has to have one of the nodes in Y_S in the position $|S|+1$, we get that $U_S = \bigcup_{y \in Y_S} U_{S,y}$. Thus, $A_S = \min_{y \in Y_S} \min_{u \in U_{S,y}} \mathrm{swap}(u,v)$.

Then, as in the proof of Theorem 4.3, we can show that

$$\min_{u \in U_{S,y}} \mathrm{swap}(u,v) = A_{S\cup\{y\}} + I_{y,S\cup\{y\}}.$$

To this end, take $u \in U_{S,y}$ minimizing $\mathrm{swap}(u,v)$ and observe that in u the first $|S|$ candidates are in the same order in which they appear in v. Let u' be a vote obtained from u by having the first $|S|+1$ candidates ordered according to v (i.e., candidate y is relocated). Then, u' actually minimizes $\mathrm{swap}(u,v)$ in $U_{S\cup\{y\}}$ (the first $|S|+1$ candidates are in the optimal order, and if reordering the last $m-|S|-1$ candidates was possible, it would also be possible to reorder them in that way in u decreasing the distance). Finally, $\mathrm{swap}(u,v) - \mathrm{swap}(u',v)$ is the number of candidates from S which are less preferred by v than y, which is what we store in $I_{y,S\cup\{y\}}$.

Observe that in this way, we have computed A_S for each singleton set $S = \{c\}$ with $c \in C$. In $U_{\{c\}}$ we have all votes in the domain that start with c. Thus, taking the minimum over $A_{\{c\}}$ for all $c \in C$ we get the minimum distance in question. □

Algorithm 4 Distance between a ranking and SP(G), where G is a tree

```
Input: Vote $v \in \mathcal{L}(C)$, tree $G$ with $C$ as nodes
  PHASE 1, PRECOMPUTATION:
  $\mathcal{S} = (S_1, \cdots, S_\ell) \leftarrow$ a sequence of subsets of $C$, such that:
                    $S \in \mathcal{S} \Leftrightarrow S \neq \emptyset$ and $S$ connected in $G$
                    $S_i \supsetneq S_j \Rightarrow i < j$
  for $S \in (S_\ell, S_{\ell-1}, \ldots, S_1)$ do
      $X_S \leftarrow$ leaves in graph induced by $S$
      for $x \in X_S$ do
          if $S = \{x\}$ then
              $I_{x,S} \leftarrow 0$
          else
              $y \leftarrow$ arbitrary node from $X_S \setminus \{x\}$
              $I_{x,S} = I_{x,S\setminus\{y\}} + [x \succ_v y]$
  PHASE 2, MAIN COMPUTATION:
  $A_{S_1} \leftarrow 0$
  for $S \in (S_2, \ldots, S_\ell)$ do
      $Y_S \leftarrow$ nodes in $C \setminus S$ connected to $S$
      $A_S \leftarrow \min_{y \in Y_S}(A_{S\cup\{y\}} + I_{y,S\cup\{y\}})$
  return $\min_{c \in C} A_{\{c\}}$
```

B.2 Algorithms for Group-Separable Domains

Now, let us look at the specific cases of GS/bal and GS/cat. Let as consider GS/cat first, and let v be the ranking whose swap distance from GS/cat we want to compute. We use an algorithm very similar to the general one, but processing the internal nodes in the decreasing order of their distance from the root, and using additional data structures. Namely, when we consider an internal node whose children are a leaf associated with some candidate c and a subtree whose leaves hold candidates from the set $C' = \{c'_1, \ldots, c'_t\}$, then we assume that we also have a data structure that for each $c'_i \in C$ holds the position that c'_i has in v. We require that it is possible to insert positions into this data structure in time $O(\log m)$ and that this data structure can also answer in $O(\log m)$ time how many of the positions that it stores are earlier in v than a given one (so, this data structure can be, e.g., a classic red-black tree, annotated with sizes of its subtrees [9]). Now, we can simply query the data structure for the number *inv* of candidates in D that are ranked ahead of d (i.e., whose position is smaller than $\mathrm{pos}_v(d)$). This is the number of inversions imposed by the current node in case we order its children, so that in the frontier we have $\{d\} \succ C'$. $t - inv$ is the number of inversions imposed in the reversed configuration. We implement the configuration that leads to fewer inversions (or we choose one arbitrarily in case of a tie), we insert $\mathrm{pos}_v(d)$ into the data structure, and we proceed to the parent node of the current one (or terminate, in case the current node was a root). The correctness follows from the correctness of the general algorithm. The running time follows from the fact that the tree has $O(m)$ internal nodes, and for each of them we need time $O(\log m)$.

THEOREM B.3. *There is an algorithm that computes the distance between a given vote and* GS/cat *(represented via a* GS*-tree) in time* $O(m \log m)$.

In case of GS/bal, we proceed similarly as in the classic Merge Sort algorithm. Let T be a balanced GS-tree and let v be the ranking under consideration. As above, our algorithm manipulates the ordering of the children of each node, to obtain a tree whose frontier u minimizes $\mathrm{swap}(u, v)$. We use a recursive procedure that given an internal node z with two children, z_ℓ on the left and z_r on the right, such that $A = \{a_1, \ldots, a_x\}$ is the set of candidates associated with the leaves of the tree rooted at z_ℓ and $B = \{b_1, \ldots, b_y\}$ is the set of candidates associated with the leaves of the tree rooted at z_r, proceeds as follows:

(1) It calls itself recursively on z_ℓ and z_r (unless a given subtree is a leaf). These calls order the children within the respective subtrees to minimize the number of inversions between the candidates in A and v and between the candidates in B and v. Additionally, they return rankings v_A and v_B that are equal to v restricted to A and B, respectively. Without loss of generality, we assume that v_A orders the candidates in A according to their indices, and so does v_B for the candidates in B

(2) We perform the "merge" step, to decide whether to reverse the order of children of z and to obtain $v_{A\cup B}$ (i.e., v restricted to the candidates in $A \cup B$). We first consider the case where we do not reverse the order of z's children. Initially, we set the number of inversions between to be 0 and, then, we fill-in $v_{A\cup B}$ from the top position to the bottom one, by considering the prefixes of v_A and v_B. Suppose that we have already filled-in the top $k - 1$ positions in $v_{A\cup B}$ with $i - 1$ candidates from A and $j - 1$ candidates from B. The candidate on the k-th position in $v_{A\cup B}$ will either be the i-th candidate from v_A or the j-th candidate from v_B, i.e., either a_i or b_j. If $a_i \succ_v b_j$ then we choose a_i, and otherwise we choose b_j and increase the number of inversions by $x - (i - 1)$ because, in this configuration, in the frontier of our tree b_j is ranked below $a_i, a_{i+1}, \ldots, a_x$. After we use up all the candidates of A or B, then we fill-in $v_{A\cup B}$ with those from the other set, in the order in which they appear in v_A or v_B, respectively. Let *inv* be the computed number of inversions. If we reversed the order of children of z, then then number of inversions would be $|A||B| - inv$; if this value is smaller than *inv* then we reverse the children. Finally, we output $v_{A\cup B}$.

Our algorithm executes this procedure on the root of the tree. The correctness is immediate, whereas the running time of $O(m \log m)$ follows from the fact that GS/bal trees have $O(\log m)$ levels, and on each level, the merge steps require $O(m)$ steps.

THEOREM B.4. *There is an algorithm that computes the distance between a given vote and* GS/bal *(represented via a* GS*-tree) in time* $O(m \log m)$.

B.3 Algorithms for Single-Crossing and Euclidean Domains

B.3.1 Single-Crossing Single-crossing domain contains $O(m^2)$ votes, thus computing swap distance to each of them and taking the minimum would give $O(m^3\sqrt{\log m})$ algorithm [8].

However, in practice, we often want to compute the distance from multiple given votes to a single fixed domain. For that case, we present an algorithm that needs a preprocessing step that also runs in time $O(m^3\sqrt{\log m})$ (this time due to the bottleneck in recognizing a single-crossing ordering of voters), but then, for each input vote, allows for computation of the distance in $O(m^2)$.

The preprocessing step involves sorting the votes in the domain in a sequence that witnesses the single-crossingness $(u_0, u_1, \dots, u_M)$, where $M = \binom{m}{2}$. This can be done in time $O(Mm\sqrt{\log M}) = O(m^3\sqrt{\log m})$ [2, 13]. Next, for each $i \in [M]$, we establish the unique pair of candidates $(a_i, b_i) \in C \times C$ such that $a_i \succ_{u_i} b_i$ but $b_i \succ_{u_{i-1}} a_i$. We can establish all of them in time $O(m^2 \log m)$ by looking at each pair of candidates and finding the place where its ordering switches using binary search.

Now, for each input vote $v \in \mathcal{L}(C)$ we first find the vector pos_v in which we keep the position of every candidate in C according to v. This can be computed in time $O(m \log m)$ by sorting the arguments of the list in which we store vote v. Next, we compute $\mathrm{swap}(u_0, v)$, again in time $O(m\sqrt{\log m})$ [8]. Further, for each $i \in [M]$, we check whether $\mathrm{pos}_v(a_i) < \mathrm{pos}_v(b_i)$. If it holds, then it means that in u_i candidates a_i and b_i are ordered in the same way as in v, which is the opposite ordering to that in u_{i-1}. Since all other pairs are ordered in the same way in u_i and u_{i-1}, we get that $\mathrm{swap}(u_i, v) = \mathrm{swap}(u_{i-1}, v) - 1$. If $\mathrm{pos}_v(a_i) > \mathrm{pos}_v(b_i)$ holds, then analogously $\mathrm{swap}(u_i, v) = \mathrm{swap}(u_{i-1}, v) + 1$. In this way, we can compute swap distance from v to each of $u_0, u_1, \dots, u_M$ in time $O(m^2)$. Finally, we output the minimum of these values.

B.3.2 Euclidean For Euclidean elections we proceed largely analogous to how we treated single-crossing elections. We know that there are $O(m^{2d})$ votes in the domain, where d is the dimension of the Euclidean space. Hence, the brute-force algorithm of computing the distances directly and taking the minimum would give us running time $O(m^{2d+1}\sqrt{\log m})$. However, we can find an alternative algorithm with $O(m^{2d+2})$ preprocessing step and $O(m^{2d})$ running time for each input vote.

For the preprocessing step we construct a graph in which the votes in the domain are vertices and the edge appears when the swap distance between two votes is equal to 1. Then, let $(u_0, u_1, \dots, u_M)$ be a sequence of votes we get when we run a DFS on this graph. Also, for each $i \in [M]$ let p_i denote the parent of u_i in the spanning tree that we get as a result. Then, let $(a_i, b_i) \in C \times C$ be a unique pair of candidates such that $a_i \succ_{u_i} b_i$ but $b_i \succ_{p_i} a_i$.

We construct the graph and identify the associated pair of candidates on each edge in overall time $O(m^{2d+2})$. To do so, we first organize all votes in the domain using a trie (prefix tree) in time $O(m^{2d+1})$, which enables lexicographic ordering and allows membership checks in $O(m)$ time. Next, for each vote and for each pair of consecutive candidates in a node, we check whether the vote obtained by swapping this pair belongs to the domain. DFS consider at most $O(m^{2d+1})$ many edges. Since each membership check takes $O(m)$, the total running time is $O(m^{2d+2})$.

Now, as in Appendix B.3.1 for each input vote $v \in \mathcal{L}(C)$ we first find the vector pos_v with position of every candidate in C according to v, which we compute in time $O(m \log m)$. Then, in time $O(m\sqrt{\log m})$ [8] we compute $\mathrm{swap}(u_0, v)$. Next, iteratively, for each $i \in [M]$ we check whether $\mathrm{pos}_v(a_i) < \mathrm{pos}_v(b_i)$. If yes, $\mathrm{swap}(u_i, v) = \mathrm{swap}(p_i, v) - 1$, otherwise, $\mathrm{swap}(u_i, v) = \mathrm{swap}(p_i, v) + 1$. In this way, we compute the swap distances between v and all the votes in the domain in time $O(m^{2d})$. Finally, we output the minimum of these values.

B.4 Hardness for Single-Peaked-on-a-Graph Domains

In this section, we provide a complete proof of Theorem 4.6 that finding a distance to an arbitrary single-peaked-on-a-graph domain is NP-complete.

Theorem 4.6. *Given a graph G, a vote $v \in \mathcal{L}(V(G))$, and an integer $d \in \mathbb{N}$, deciding if* $\mathrm{swap}(SP(G), v) \le d$ *is* NP*-complete.*

Proof. If we are given a ranking $u \in SP(G)$ that is the closest to the given vote, v, then checking if $\mathrm{swap}(u, v) \le d$ can be done in polynomial time. Thus, the problem belongs to NP. Hence, in the remainder of the proof, we focus on showing the hardness.

To this end, we will provide a reduction from SetCover. In this problem, we are given a universe of elements $\mathcal{U} = \{u_1, \dots, u_n\}$, a family of $\mathcal{U}$'s subsets $\mathcal{S} = \{S_1, \dots, S_m\}$, and an integer $k \in \mathbb{N}$. The question is whether there exists a subset $K \subseteq \mathcal{S}$ of size $|K| = k$, known as a *set cover*, that contains all elements from the universe, i.e., $\bigcup_{S_j \in K} S_j = \mathcal{U}$. Answering this question is known to be NP-complete [22]. Without loss of generality, we assume that $n > 2$ and $m > k$.

For each instance of SetCover we construct an instance of our problem as follows (see Figure 4 for an illustration). We let the set of candidates $C = V(G)$ contain three groups of candidates: (1) *$n{\cdot}m$ element candidates* $(c_{i,j})_{i\in[n], j\in[m]}$, among which n candidates, $(c_{i,1})_{i\in[n]}$, are called *first element candidates*; (2) *m subset candidates* $s_1, \dots, s_m$; and (3) $n^2 \cdot m^2 + 1$ *path candidates* $p_0, p_1, \dots, p_{n^2\cdot m^2}$. As for the edges in graph G, for each $i \in [n]$, we connect all element candidates $c_{i,1}, \dots, c_{i,m}$, to form a path, and the same we do with all of the path candidates $p_0, p_1, \dots, p_{n^2\cdot m^2}$. Additionally, we connect p_0 to all set candidates $s_1, \dots, s_m$. Finally, for each $j \in [m]$, we connect set candidate s_j to the first element candidates with indices corresponding to the indices of the elements of subset S_j. Formally,

$$\begin{aligned} E(G) = {} & \{\{c_{i,j}, c_{i,j+1}\} : i \in [n], j \in [m-1]\} \\ & \cup \{\{p_{i-1}, p_i\} : i \in [n^2 \cdot m^2]\} \\ & \cup \{\{p_0, s_j\} : j \in [m]\} \\ & \cup \{\{s_j, c_{i,1}\} : j \in [m], u_i \in S_j\}. \end{aligned}$$

In the given input vote v, the top candidate is p_0, followed by all element candidates, then remaining path candidates, and lastly the subset candidates at the bottom of the ranking, i.e.,

$$\begin{aligned} & p_0 \succ_v c_{1,1} \succ_v c_{1,2} \succ_v \cdots \succ_v c_{1,m} \succ_v \\ & c_{2,1} \succ_v \cdots \succ_v c_{n-1,m} \succ_v c_{n,1} \succ_v c_{n,2} \succ_v \cdots \succ_v c_{n,m} \succ_v \\ & p_1 \succ_v p_2 \succ_v \cdots \succ_v p_{n^2\cdot m^2} \succ_v s_1 \succ_v s_2 \succ_v \cdots \succ_v s_m. \end{aligned}$$

Finally, we set $d = (k+1)n^2 \cdot m^2 - 1$.

First, let us show that if there exists a set cover K in the original instance, then $\mathrm{swap}(\mathrm{SP}(G), v) \le d$. Let K' contain all the subset candidates corresponding to subsets in K, i.e., $K' = \{s_j : S_j \in K\}$. Then, let u be a vote obtained from v by moving all subset candidates in K' upwards in the ranking so that all of them are between candidates p_0 and $c_{1,1}$ (the ordering of the remaining candidates is the same). For each $s_j \in K'$, there are exactly $n \cdot m$ element candidates, $n^2 \cdot m^2$ path candidates, and at most m subset

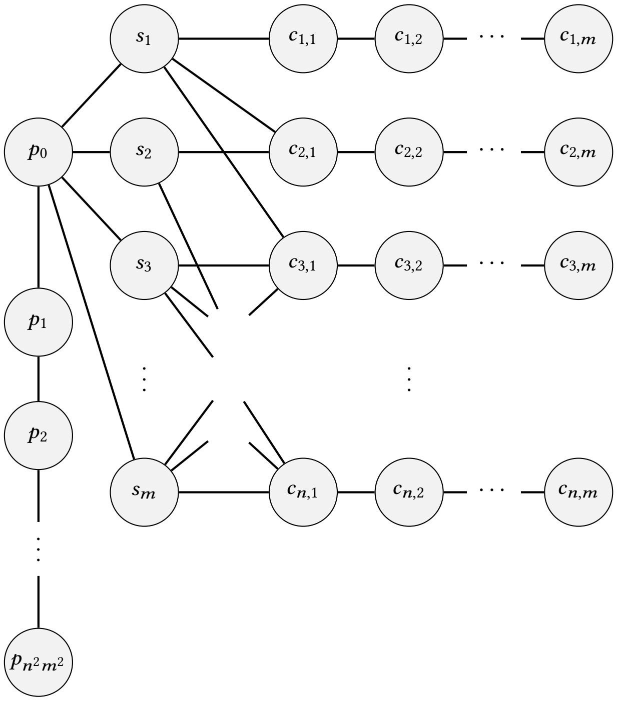

Figure 4: An illustration of the construction from the proof of Theorem 4.6.

candidates for which the ordering in u and v is different. Hence,

$$\begin{aligned}\mathrm{swap}(u, v) &\leq kn^2m^2 + knm + km\\ &= kn^2m^2 + km(n+1)\\ &\leq (k+1)n^2m^2 - 1\\ &= d,\end{aligned}$$

where the last inequality comes from our assumption that $n \geq 3$ and $m > k$. Moreover, we can observe that u belongs to the $\mathrm{SP}(G)$ domain. Indeed, for each $t \in [k+1]$, the first t candidates in u form a connected subgraph in G, as each subset candidate is connected to p_0. Then, for each $t \in \{k+2, \ldots, k+1+n\cdot m\}$, each element candidate $c_{i,j}$ for $i \in [n]$, $j \in [m]$ is connected through the path of element candidates (that all appear before it in u) to the first element candidate $c_{i,1}$, which in turn is connected to some $s_j \in K'$ (as there is $S_j \in K$ such that $u_i \in S_j$). Finally, for $t > k+1+n\cdot m$, every path candidate p_i for $i \in [n^2m^2]$ is connected to p_0 through a path of path candidates (that all appear before it in u), and every set candidate $s_j \notin K'$ is connected directly to p_0. Therefore, indeed, $\mathrm{swap}(\mathrm{SP}(G), v) \leq \mathrm{swap}(u, v) \leq d$.

In the remainder of the proof, let us assume that there is no set cover in the original SetCover instance and let us show that this implies that $\mathrm{swap}(\mathrm{SP}(G), v) > d$. Take an arbitrary vote $u \in \mathrm{SP}(G)$. Let $i^* \in [n]$ be such that $c_{i^*,1}$ is the least preferred among the first element candidates in u, i.e., $c_{i,1} \succ_u c_{i^*,1}$, for each $i \in [n] \setminus \{i^*\}$.

Observe that it must hold that $c_{i^*,1} \succ_u c_{i^*,2} \succ_u \cdots \succ_u c_{i^*,m}$ (otherwise, if there is $j < j' \leq m$ such that $c_{i^*,j'} \succ_u c_{i^*,j}$, then the subset of the first t candidates up to $c_{i^*,j'}$ is not connected in G, as $c_{i^*,j}$ is on the only path from $c_{i^*,j'}$ to $c_{i,1}$ for any $i \in [n] \setminus \{i^*\}$). Moreover, by the definition of $\mathrm{SP}(G)$ domain, the set of candidates that are weakly preferred over $c_{i^*,1}$ in u, i.e., $C' = \{c \in C : c \succ_u c_{i^*,1}\} \cup \{c_{i^*,1}\}$, must form a connected subgraph in G. This means that C' must contain at least one subset candidate connected to each first element candidate. Let $K' = \{s_1, \ldots, s_m\} \cap C'$ be a subset of all subset candidates in C' and let $K = \{S_j : s_j \in K'\}$ be a set of corresponding subsets in $\mathcal{S}$. We know that K covers all the elements in $\mathcal{U}$, but since there is no set cover of size k, in the SetCover instance, we get that $|K'| = |K| \geq k+1$.

Now, let P denote the set of path candidates, excluding p_0 that are ranked above $c_{i^*,1}$ in u, i.e., $P = C' \cap \{p_1, \ldots, p_{n^2\cdot m^2}\}$. Then, there are at least $m \cdot |P|$ pairs of an element candidate from the path $(c_{i^*,j})_{j\in[m]}$ and a path candidate in P that are ordered differently in u and v. Moreover, there are at least $(k+1)\cdot(n^2\cdot m^2 - |P|)$ pairs of a subset candidate in K' and a path candidate in $\{p_1, \ldots, p_{n^2\cdot m^2}\} \setminus P$ that are ordered differently in u and v. Hence,

$$\begin{aligned}\mathrm{swap}(u, v) &\geq m\cdot|P| + (n^2\cdot m^2 - |P|)\cdot(k+1)\\ &\geq (k+1)\cdot|P| + (n^2\cdot m^2 - |P|)\cdot(k+1)\\ &= n^2\cdot m^2\cdot(k+1)\\ &> d,\end{aligned}$$

where the second inequality comes from our assumption that $m \geq k+1$. This concludes the proof. □

C Missing Proofs for Section 5

Proposition 5.1. *Let D be the* GS/bal *domain for $m = 2^k$ candidates. For every ranking $v \in D$ there are exactly $2^{k-1} - 1$ unique ones from $\mathcal{L}(C) \setminus D$ at swap distance 1 from v.*

Proof. Let $m = 2^k$ be the number of candidates, let the candidate set be $C = \{c_1, \ldots, c_m\}$, and let D be our GS/bal domain for C. Further, let v be ranking in $\mathcal{L}(C)$. W.l.o.g., we can assume that v ranks the candidates as $c_1 \succ c_2 \succ \cdots \succ c_m$. For each $i \in [m-1]$, let $v(i)$ be the ranking obtained from v by swapping candidates c_i and c_{i+1}. These are all the rankings from $\mathcal{L}(C)$ that are at swap distance 1 from v. By definition of D, for every odd $i \in [m-1]$, $v(i)$ is in D, and for each even i it is in $\mathcal{L}(C) \setminus D$. Further, for each even i, swap distance of $v(i)$ to every member of D other than v is larger than 1: Indeed, in every vote from D, c_i and c_{i-1} must be ranked next to each other. To achieve this, by performing a single swap on $v(i)$, we need to swap c_{i+1} with c_{i-1} or c_i. The former, does not lead to a vote from D, the latter leads to v. This completes the proof. □

Proposition 5.2. *Consider* GS/cat *over $m \geq 4$ candidates. For every ranking $v \in$* GS/cat *there are exactly $m-3$ unique ones from $\mathcal{L}(C) \setminus D$ at swap distance 1 from v, and one ranking from $\mathcal{L}(C)$ that is at swap distance 1 from v and one other ranking in* GS/cat.

Proof. Let T be a binary caterpillar tree with c_1 being the leaf closest to the root, followed by c_2, and so on up to c_m. Let D be a GS/cat domain consistent with tree T.

Observe that in each vote $v \in D$, when we read the candidates along $\succ_v$, the indices are increasing until candidate c_m, after which they are decreasing.[2] After swapping any pair of candidates in v, except for the pair $\{c_{m-1}, c_m\}$, we obtain v' that does not have this

[2] This property makes GS/cat in some sense dual to SP, which was observed e.g. in [6].

property, hence $v' \notin D$. Moreover, unless we swapped c_{m-2} with either c_m or c_{m-1}, the only way to restore this property by a single swap, is to go back to v. Thus, $\mathrm{swap}(v', u) > 1$, for each $u \in D \setminus \{v\}$.

On the other hand, if v' is obtained from v by swapping c_{m-2} with c_m or c_{m-1} (whichever is adjacent to c_{m-2} in v), then in v', candidate c_{m-2} is ranked exactly between c_m and c_{m-1}. Swapping it with any of these two candidates yields a vote from D (and no single swap apart from these two results in that). □

Proposition 5.3. *Let $D = \mathrm{GS}(T)$ be a group separable domain over candidate set C. Then, for each $v \in \mathrm{GS}(T)$, $\mathrm{npop}(v) = 1$.*

Proof. For a contradiction, assume that the thesis does not hold. Then, there exist $u, v \in D$ such that $\mathrm{npop}(u) > \mathrm{npop}(v)$. Let $\pi : C \to C$ be a permutation such that $v = \pi(u)$, where $\pi(u)$ denotes a vote in which each candidate $c \in C$ is replaced by $\pi(c)$. In this way, by a slight abuse of notation, π is also a permutation of $\mathcal{L}(C)$.

By the definition of GS domain, π corresponds to rotating the children of certain internal nodes in T. Thus, for every $w \in \mathcal{L}(C)$ it holds that $w \in D$ if and only if $\pi(w) \in D$. Moreover, for each $w, w' \in \mathcal{L}(C)$ we have that $\mathrm{swap}(w, w') = \mathrm{swap}(\pi(w), \pi(w'))$. Both facts imply that $\mathrm{npop}(u) = \mathrm{npop}(\pi(u))$, as $\mathrm{npop}(\cdot)$ is invariant under π (since π does not affect the domain, nor the swap distance). However, this leads to a contradiction as $\mathrm{npop}(u) = \mathrm{npop}(\pi(u)) = \mathrm{npop}(v) < \mathrm{npop}(u)$. □

Proposition 5.4. *Let us fix value k and let $D_2, D_3, \ldots$ be a sequence of domains, where each D_m contains at most k rankings over m candidates. Then $\lim_{m\to\infty} \text{out-div}(D_m) = 0$.*

Proof. Let UN_m denote the UN election with m candidates. As already noted, the average normalized swap distance of a domain D is equal to the normalized Kemeny score of D with respect to the UN_m election, i.e., $m!\binom{m}{2} \cdot \mathrm{ansd}(D) = \mathrm{kem}_{\mathrm{UN}_m}(D)$. This, in turn, is not smaller than the k-Kemeny score of the UN_m election, where $k = |D|$, which gives us $m!\binom{m}{2} \cdot \mathrm{ansd}(D) \geq k\text{-kem}(\mathrm{UN}_m)$. This yields the following bound on the outer diversity:

$$\text{out-div}(D) = 1 - 2 \cdot \mathrm{ansd}(D) \leq 1 - 2 \cdot \frac{k\text{-kem}(\mathrm{UN}_m)}{m!\binom{m}{2}}.$$

Faliszewski et al. [19, Proposition 3.6] showed that for every $k \in \mathbb{N}$, it holds that

$$\lim_{m\to\infty} \frac{k\text{-kem}(\mathrm{UN}_m)}{\frac{1}{2} \cdot m!\binom{m}{2}} = 1.$$

Thus,

$$\lim_{m\to\infty} \text{out-div}(D_m) \leq 1 - \lim_{m\to\infty} \frac{k\text{-kem}(\mathrm{UN}_m)}{\frac{1}{2} \cdot m!\binom{m}{2}} = 0.$$

□

D Most Diverse Domains

Below, we provide a formal definition for the Most Diverse Domain.

Definition D.1. For a set of candidates C and an integer $k \leq |C|!$ the Most Diverse Domain problem asks for a set $D \subseteq \mathcal{L}(C)$ of size k that maximizes $\text{out-div}(D)$.

We observe that an optimal solution to Most Diverse Domain is a set of k rankings that achieves the optimal k-Kemeny score for the election $(C, \mathcal{L}(C))$. Moreover, finding k rankings that realize the optimal k-Kemeny score of $(C, \mathcal{L}(C))$ can be formulated as the classic clustering problem k-Median of the metric space of all possible rankings together with the swap distance, i.e., $(\mathcal{L}(C), \mathrm{swap})$.

To compute optimal solutions for Most Diverse Domain, we used a standard Integer Linear Program (ILP) for k-Median. Unfortunately, this approach is computationally expensive because the ILP has size $\Theta((m!)^2)$. A faster, heuristic alternative is simulated annealing: We initialize a random set of k rankings and iteratively attempt to improve the solution by replacing a single ranking to reduce the total swap distance. This heuristic appears surprisingly effective, likely because randomly sampling k rankings from the impartial culture model already yields near-optimal solutions, especially for large k.

For completeness, we provide an ILP formulation for Most Diverse Domain below that is equivalent to an ILP for k-Median in a specific metric space $(\mathcal{L}(C), \mathrm{swap})$ and with a specific set of points to cluster $\mathcal{L}(C)$.

Let $\mathcal{L}(C) = \{u_1, \ldots, u_{m!}\}$ be a set of rankings over a set C of m candidates, and let k denote the size of domain. For readability, we define $n = m!$. For each ranking $u_i \in \mathcal{L}(C)$, we define a binary variable y_i with the intention that value 1 indicates that ranking u_i is selected to a solution. For each pair of rankings $u_i, u_j \in \mathcal{L}(C)$, we define a binary variable x_{ij} with the intention that value 1 means that a ranking u_i has u_j as the closest ranking in a solution (u_j is a representative, or cluster center, of u_i). Let d_{ij} denote the swap distance between rankings u_i and u_j, i.e., $d_{ij} = \mathrm{swap}(u_i, u_j)$. We introduce the following constraints:

$$\begin{aligned} x_{ij}, y_i &\in \{0,1\}, && \forall i, j \in [n] \\ \textstyle\sum_{i\in[n]} x_{ij} &= 1, && \forall j \in [n] && (1) \\ x_{ij} &\leq y_i, && \forall i, j \in [n] && (2) \\ \textstyle\sum_{i\in[n]} y_i &= k. && && (3) \end{aligned}$$

Constraint (1) ensures that each ranking is assigned to exactly one selected ranking. Constraint (2) ensures that a vote can only be assigned to another vote if that vote is selected. Constraint (3) ensures that exactly k rankings are selected. The objective function defined in (4) minimizes the total cost, i.e., the total swap distance:

$$\min \sum_{i\in[n]} \sum_{j\in[n]} d_{ij} \cdot x_{ij}. \qquad (4)$$

D.1 Computing Most Diverse Domain

Our simulated annealing algorithm operates as follows. We begin with a randomly generated set of rankings. At each iteration, we uniformly at random remove one of the rankings and add one ranking sampled from IC. If the new solution is better than the current one, it is always accepted. Otherwise, it is accepted with probability

$$P = \exp\left(\frac{E_{\text{new}} - E_{\text{current}}}{T}\right),$$

where T denotes the current temperature. The initial temperature is set to $T_0 = 0.5$, and it decreases geometrically with a cooling rate of 0.95 per iteration. Moreover, we perform at most 256 iterations.

In Figure 5, we compare the performance of simulated annealing and ILP for the case of six candidates. As shown, the solution found

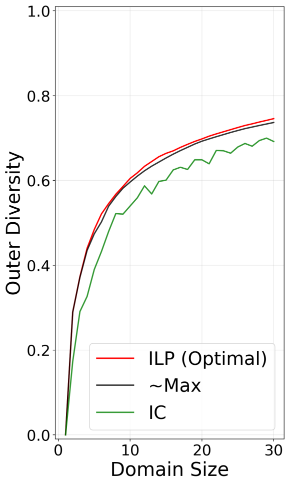

Figure 5: Comparison of the optimal diversity (red line) and the one achieved by simulated annealing (black line) for 6 **candidates.**

by simulated annealing is nearly optimal. Moreover, note that simply sampling votes from the IC distribution serves as an effective heuristic.

D.2 Largest Gap in a Domain

A domain can be considered diverse if it is well distributed across a metric space of all possible rankings, i.e., $(\mathcal{L}(C), \text{swap})$. This implies that there are no large gaps between rankings within the domain. Consequently, it is natural to search for the largest such gap. To formalize this, we define a decision problem of finding the center of a ball in $(\mathcal{L}, \text{swap})$ with a given radius that contains no rankings from the given domain D.

Definition D.2. In the Farthest Permutation problem we are given $D \subseteq \mathcal{L}(C)$ and $r_{\text{far}} \in \{-1, 0\} \cup \mathbb{N}$. We ask if there exists a ranking $f \in \mathcal{L}(C)$ which swap distance to D is at least r_{far}, i.e.,

$$r_{\text{far}} < \min_{v \in D} \text{swap}(v, f) = \text{swap}(D, f).$$

We emphasize that the definition uses a strict inequality because the goal is to identify a ball that excludes all rankings from D. For example, if $D = \mathcal{L}(C)$ then the only value of r_{far} for which a response is YES, is -1. For $D \subset \mathcal{L}(C)$, every $f \in \mathcal{L}(C) \setminus D$ is at distance 1 from D, so $r_{\text{far}} \geq 0$ in this case. We can also define an optimization version of Farthest Permutation, searching for a maximum r_{far} for which a response is YES.

In the subsequent proofs, we will rely on results concerning the Kemeny 1-Center problem, which may be regarded as a dual problem to Farthest Permutation in the sense that Farthest Permutation looks for a ball of radius r_{far} where none of domain rankings are included, but Kemeny 1-Center looks for a ball of radius r_{center} where all of domain rankings are included.

Definition D.3. In the Kemeny 1-Center problem we are given $D \subseteq \mathcal{L}(C)$ and $r_{\text{center}} \in \{0\} \cup \mathbb{N}$. We ask if there exists a ranking $c \in \mathcal{L}(C)$ which swap distance to every element in D is at most r_{center}, i.e.,

$$\max_{v \in D} \text{swap}(v, c) \leq r_{\text{center}}.$$

The duality mentioned can be formalized in a quantitative way as done in Lemma D.4 which essentially says that for every permutation $x \in \mathcal{L}(C)$, the sum of radii of two balls: 1) a ball with a Farthest Permutation objective and a center in x and; 2) a ball with a Kemeny 1-Center objective and a center in $\text{rev}(x)$, where $\text{rev}(x)$ is a reversed permutation of x; is always equal to $\binom{m}{2} - 1$, i.e., a maximum distance between two permutations of m elements decreased by 1.

Formally, for $D \subseteq \mathcal{L}(C)$ and $x \in \mathcal{L}(C)$ we define the radii described above as: $\text{FP}(D, x) = \text{swap}(D, x) - 1$ and $\text{K1C}(D, x) = \max_{v \in D} \text{swap}(v, x)$.

Lemma D.4. *For every $D \subseteq \mathcal{L}(C)$ and every $x \in \mathcal{L}(C)$ we have*

$$\text{FP}(D, x) + \text{K1C}(D, \text{rev}(x)) = \binom{m}{2} - 1.$$

Proof. Let us fix $D \subseteq \mathcal{L}(C)$ and $x \in \mathcal{L}(C)$. First we observe that, by the swap distance definition, we have

$$\text{swap}(v, x) + \text{swap}(v, \text{rev}(x)) = \binom{m}{2}.$$

Using it, we obtain a sequence of equalities:

$$\begin{aligned}
\text{FP}(D, x) &= -1 + \min_{v \in D} \text{swap}(v, x) \\
&= -1 + \min_{v \in D} \left(\binom{m}{2} - \text{swap}(v, \text{rev}(x)) \right) \\
&= \binom{m}{2} - 1 - \max\nolimits_{v \in D} \text{swap}(v, \text{rev}(x)) \\
&= \binom{m}{2} - 1 - \text{K1C}(D, \text{rev}(x)).
\end{aligned}$$

This finishes the proof. □

The lemma implies that hardness of finding a solution to Kemeny 1-Center implies hardness of finding a solution to Farthest Permutation as well as an additive approximation algorithm with additive loss guarantee of at most β for Kemeny 1-Center is also an approximation algorithm for Farthest Permutation with the same additive loss guarantee. The two results results are formally presented in the following two theorems. We observe that Theorem D.5 directly implies the result stated in Theorem 6.1.

Theorem D.5. *Farthest Permutation is NP-complete, even when $|D| = 4$.*

Proof. The inclusion in NP is straightforward as this is enough to compute all pairwise distances between a solution and elements of a domain and check if any of them is equal or smaller than r.

In order to show NP-hardness for $|D| = 4$, we will construct a reduction from Kemeny 1-Center which is NP-hard for $|D| = 4$, where all input orders are distinct (see [11] for the original proof and [4, Theorem 5] for its correction).

Let $D \subseteq \mathcal{L}(C), r_{\text{center}} \in \{0\} \cup \mathbb{N}$ be an input of Kemeny 1-Center.[3] We define an input of Farthest Permutation simply by providing the same domain D and $r_{\text{far}} = \binom{m}{2} - 1 - r_{\text{center}}$.

Correctness of the reduction directly follows from Lemma D.4. For completes we provide the two formal implications below.

[3] While the original definition of Kemeny 1-Center allows inputs with non-distinct orders, every such instance can, without loss of generality, be reduced to an equivalent instance consisting solely of distinct orders.

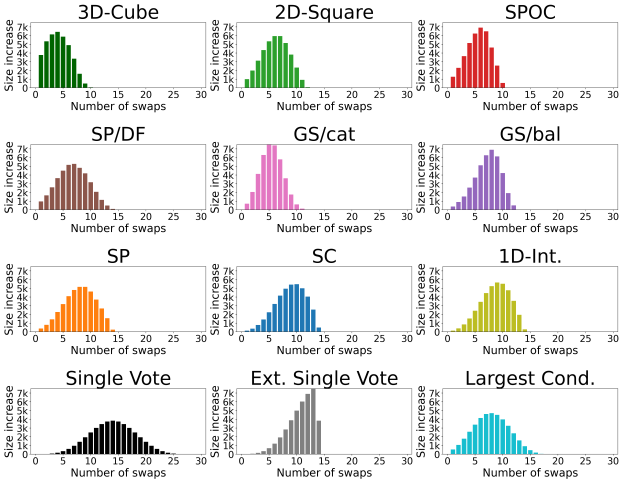

Figure 6: Histograms of votes at a given swap distance.

(⇒) If (D, r_{center}) is a YES-instance of Kemeny 1-Center then there exists $c \in \mathcal{L}(C)$ such that $\text{K1C}(D, c) \leq r_{\text{center}}$ and we obtain

$$\begin{aligned} \text{swap}(D, \text{rev}(c)) &= \text{FP}(D, \text{rev}(c)) + 1 \\ &\overset{\text{Lemma D.4}}{=} \binom{m}{2} - 1 - \text{K1C}(D, c) + 1 \\ &\geq \binom{m}{2} - r_{\text{center}} > r_{\text{far}}. \end{aligned}$$

Therefore, $\text{rev}(c)$ is a solution to the Farthest Permutation instance (D, r_{far}).

(⇐) If (D, r_{far}) is a YES-instance of Farthest Permutation then, analogously, there exists $f \in \mathcal{L}(C)$ such that $\text{FP}(D, f) \geq r_{\text{far}}$ and we obtain

$$\begin{aligned} \max_{v \in D} \text{swap}(v, \text{rev}(f)) &= \text{K1C}(D, \text{rev}(f)) \\ &\overset{\text{Lemma D.4}}{=} \binom{m}{2} - 1 - \text{FP}(D, f) \\ &\leq \binom{m}{2} - 1 - r_{\text{far}} = r_{\text{center}}. \end{aligned}$$

Therefore, $\text{rev}(f)$ is a solution to the Kemeny 1-Center instance (D, r_{center}). This finishes the proof. □

The duality between Farthest Permutation and Kemeny 1-Center presented in Lemma D.4 holds for centers of balls at x and $\text{rev}(x)$. The duality can be also expressed,in Lemma D.6, in terms of how far radii of balls with centers at x and $\text{rev}(x)$ are from optimum solutions. For that we will need a few more definitions. For a given $D \subseteq \mathcal{L}(C)$, let $\text{FP}(D)$ be a maximum r_{far} for which (D, r_{far}) is a YES-instance of Farthest Permutation. Analogously, let $\text{K1C}(D)$ be a minimum r_{center} for which (D, r_{center}) is a YES-instance of Farthest Permutation.

Lemma D.6. *For every $D \subseteq \mathcal{L}(C), x \in \mathcal{L}(C)$ and $\beta \in \mathbb{N}$ we have*

$$\text{FP}(D, x) \geq \text{FP}(D) - \beta \Leftrightarrow \text{K1C}(D, \text{rev}(x)) \leq \text{K1C}(D) + \beta.$$

Proof. We fix $D \subseteq \mathcal{L}(C), x \in \mathcal{L}(C)$ and $\beta \in \mathbb{N}$. Let x_{far} be such that $\text{FP}(D, x_{\text{far}}) = \text{FP}(D)$. Then, by Lemma D.4, we have that $\text{K1C}(D, \text{rev}(x_{\text{far}})) = \text{K1C}(D)$. We obtain a sequence of equivalent inequalities:

$$\begin{aligned} \text{FP}(D, x) &\geq \text{FP}(D) - \beta \\ \text{FP}(D, x) &\geq \text{FP}(D, x_{\text{far}}) - \beta \\ \binom{m}{2} - 1 - \text{K1C}(D, \text{rev}(x)) &\geq \binom{m}{2} - 1 - \text{K1C}(D, \text{rev}(x_{\text{far}})) - \beta \\ -\text{K1C}(D, \text{rev}(x)) &\geq -\text{K1C}(D) - \beta \\ \text{K1C}(D, \text{rev}(x)) &\leq \text{K1C}(D) + \beta, \end{aligned}$$

where the second equivalence comes from Lemma D.4. This finishes the proof. □

An algorithm for Farthest Permutation is an (additive) β-approximation if for an input D it outputs $x \in \mathcal{L}(C)$ such that $\text{FP}(D, x) \geq \text{FP}(D) - \beta$. An algorithm for Kemeny 1-Center is an (additive) β-approximation if for an input D it outputs $x \in \mathcal{L}(C)$ such that $\text{K1C}(D, x) \leq \text{K1C}(D) + \beta$. The following corollary is an implication of Lemma D.6.

Corollary D.7. *For a given $D \subseteq \mathcal{L}(C)$ it holds:*

(1) *Let x be an output of an additive β-approximation algorithm for Kemeny 1-Center on D. Then, $\text{rev}(x)$ is an additive β-approximate solution to Farthest Permutation on D.*
(2) *Let x be an output of an additive β-approximation algorithm for Farthest Permutation on D. Then, $\text{rev}(x)$ is an additive β-approximate solution to Kemeny 1-Center on D.*

E Additional Plots

Additional histograms of swap distances are shown in Figure 6.