Mathematics of Heterogeneous Distance Metrics

1 Simulator

This section describes the simulator as implemented in the codebase. All quantities and procedures discussed below correspond precisely to the computations performed: geometry sampling, heterogeneous distance perception, utility formation, ballot formation, voting rules, manipulation (optional), and evaluation metrics.

2 Basic objects and indexing

Fix:

- number of independent election instances ("profiles") $P \in \mathbb{N}$,
- number of voters $N \in \mathbb{N}$,
- number of candidates $M \in \mathbb{N}$, $M \geq 2$,
- spatial dimension $d \in \mathbb{N}$, $d \geq 1$,
- geometry bounds $p_{\min} < p_{\max}$ (defaults $p_{\min} = 0, p_{\max} = 1$),
- a random seed s (optional; if absent, NumPy generates a fresh seed),
- a numerical tolerance ε (stored in config but not used in ranking conversion).

Indices:

$$p \in \{1, \ldots, P\}, \quad v \in \{1, \ldots, N\}, \quad c \in \{1, \ldots, M\}, \quad k \in \{1, \ldots, d\}.$$

For each profile p, the simulator samples:

$$x_v^{(p)} \in [p_{\min}, p_{\max}]^d \quad \text{(voter positions)}, \qquad y_c^{(p)} \in [p_{\min}, p_{\max}]^d \quad \text{(candidate positions)}.$$

Given positions, it computes:

$$D_{v,c}^{(p)} \in \mathbb{R}_{\geq 0} \quad \text{(distance)}, \qquad u_{v,c}^{(p)} \in [0, 1] \quad \text{(utility)}, \qquad R_v^{(p)} \in \{0, \ldots, M-1\}^M \quad \text{(ranking permutation)}.$$

Here $R_v^{(p)}[0]$ is voter v's top-ranked candidate index, etc.

3 Randomness and reproducibility

All sampling uses NumPy's `default_rng`. When a seed s is provided, the same seed is used for:

- geometry sampling (voter/candidate positions),
- voting-rule tie-breaking when enabled (see Section 9.1),
- manipulation selection and poll noise when manipulation is enabled.

4 Spatial geometry generation

The simulator supports the following *implemented* geometry methods for each profile p.

4.1 Uniform hypercube (`uniform`)

Independently for all p, v, k:

$$(x_v^{(p)})_k \sim \mathrm{Unif}(p_{\min}, p_{\max}), \qquad (y_c^{(p)})_k \sim \mathrm{Unif}(p_{\min}, p_{\max}).$$

4.2 Clustered (`clustered`)

Let $\phi \in [0,1]$ be a dispersion parameter and let $\sigma_0 > 0$ be the base cluster variance (code default $\sigma_0 = 0.15$). Define:

$$\sigma_{\text{voter}} \coloneqq \phi\sqrt{d}\,\sigma_0, \qquad \sigma_{\text{cand}} \coloneqq \tfrac{1}{2}\sigma_{\text{voter}}.$$

For each profile p, sample a cluster center $m^{(p)} \in [p_{\min}+0.3\Delta,\, p_{\max}-0.3\Delta]^d$ where $\Delta = p_{\max}-p_{\min}$, and then:

$$x_v^{(p)} \sim \mathcal{N}\Big(m^{(p)}, \sigma_{\text{voter}}^2 I_d\Big) \text{ clipped coordinate-wise to } [p_{\min}, p_{\max}],$$

$$y_c^{(p)} \sim \mathcal{N}\Big(m^{(p)}, \sigma_{\text{cand}}^2 I_d\Big) \text{ clipped coordinate-wise to } [p_{\min}, p_{\max}].$$

4.3 Single-peaked 1D (`single_peaked`)

This method *forces* $d = 1$. Voters are uniform on $[p_{\min}, p_{\max}]$. Candidates are placed deterministically at evenly spaced positions between $p_{\min} + 0.2\Delta$ and $p_{\max} - 0.2\Delta$:

$$y_c^{(p)} = \big(p_{\min} + 0.2\Delta\big) + \frac{c-1}{M-1}\big(0.6\Delta\big), \qquad c = 1, \ldots, M.$$

4.4 Polarized (`polarized`)

Let $\Delta = p_{\max} - p_{\min}$. Define three centers:

$$\ell = (p_{\min} + 0.15\Delta)\mathbf{1}_d, \quad r = (p_{\max} - 0.15\Delta)\mathbf{1}_d, \quad m = \tfrac{1}{2}(p_{\min} + p_{\max})\mathbf{1}_d,$$

and a common dispersion:

$$\sigma = 0.08\,\Delta\,\sqrt{d}.$$

For each profile p, sample faction sizes:

$$N_L = \lfloor N\,U_L \rfloor, \quad U_L \sim \mathrm{Unif}(0.35, 0.50),$$

$$N_R = \lfloor N\,U_R \rfloor, \quad U_R \sim \mathrm{Unif}(0.35, 0.50),$$

$$N_M = N - N_L - N_R.$$

Then sample N_L voters from $\mathcal{N}(\ell, \sigma^2 I_d)$, N_R voters from $\mathcal{N}(r, \sigma^2 I_d)$, and N_M voters from $\mathcal{N}(m, \sigma^2 I_d)$, each clipped to $[p_{\min}, p_{\max}]^d$, and finally uniformly shuffle the voter order.

Candidates: if $M \geq 3$, then

$$y_1^{(p)} \sim \mathcal{N}(\ell, (\sigma/2)^2 I_d), \quad y_2^{(p)} \sim \mathcal{N}(m, (\sigma/2)^2 I_d), \quad y_3^{(p)} \sim \mathcal{N}(r, (\sigma/2)^2 I_d),$$

each clipped to $[p_{\min}, p_{\max}]^d$. Any candidates $c \geq 4$ are uniform in the hypercube. If $M < 3$, all candidates are uniform.

4.5 One-dimensional uniform (`1d`)

This method forces $d = 1$. Voters and candidates are uniform on $[p_{\min}, p_{\max}]$.

4.6 Two-dimensional triangle (`2d`)

This method forces $d = 2$. Voters are uniform on $[p_{\min}, p_{\max}]^2$. If $M \geq 3$, candidates $1, 2, 3$ are placed deterministically at:

$$y_1^{(p)} = \big(p_{\min}{+}0.2\Delta,\ p_{\min}{+}0.3\Delta\big), \quad y_2^{(p)} = \big(p_{\max}{-}0.2\Delta,\ p_{\min}{+}0.3\Delta\big), \quad y_3^{(p)} = \big(\tfrac{1}{2}(p_{\min}{+}p_{\max}),\ p_{\max}{-}0.2\Delta\big),$$

and any candidates $c \geq 4$ are uniform. If $M < 3$, all candidates are uniform in $[p_{\min}, p_{\max}]^2$.

5 Distance metrics

Given a voter position $x \in \mathbb{R}^d$ and candidate position $y \in \mathbb{R}^d$, the simulator implements:

$$\begin{aligned}
d_{L2}(x, y) &\coloneqq \|x - y\|_2, \\
d_{L1}(x, y) &\coloneqq \|x - y\|_1 = \sum_{k=1}^{d} |x_k - y_k|, \\
d_{\infty}(x, y) &\coloneqq \|x - y\|_\infty = \max_k |x_k - y_k|, \\
d_{\cos}(x, y) &\coloneqq 1 - \operatorname{clip}\left(\frac{\langle x, y \rangle}{(\|x\|_2 + \delta)(\|y\|_2 + \delta)},\ -1,\ 1 \right),
\end{aligned}$$

with a small constant $\delta = 10^{-10}$ used to avoid division by zero, and clip denoting truncation to $[-1, 1]$.

6 Heterogeneous distance perception

The *heterogeneous distance* option assigns a metric μ_v to each voter v as a function of that voter's spatial *centrality*. Distances are then computed using the assigned metric:

$$D_{v,c}^{(p)} \coloneqq d_{\mu_v^{(p)}}\left(x_v^{(p)},\ y_c^{(p)} \right).$$

When heterogeneous distance is disabled, a single metric μ is used for all voters:

$$D_{v,c}^{(p)} \coloneqq d_{\mu}\left(x_v^{(p)},\ y_c^{(p)} \right).$$

6.1 Centrality normalization

Let the hypercube center be:

$$z \coloneqq \tfrac{1}{2}(p_{\min} + p_{\max})\mathbf{1}_d,$$

and let the maximum possible Euclidean distance from z to any corner be the half-diagonal:

$$L_{\max} \coloneqq \sqrt{d}\,\frac{p_{\max} - p_{\min}}{2}.$$

The simulator defines each voter's centrality as normalized Euclidean distance to z:

$$\mathrm{cent}(x) \coloneqq \mathrm{clip}\left(\frac{\|x - z\|_2}{L_{\max}},\ 0,\ 1\right).$$

Note that this normalization is always Euclidean, even if the voter will later use a non-Euclidean metric for candidate evaluation.

6.2 Strategy A: center–extreme (`center_extreme`)

Parameters:

- center metric μ_{center} (default $L2$),
- extreme metric μ_{extreme} (default cosine),
- threshold $t \in [0, 1]$ (default $t = 0.5$).

For a voter at position x, define:

$$\mu(x) \coloneqq \begin{cases} \mu_{\text{center}}, & \mathrm{cent}(x) \le t, \\ \mu_{\text{extreme}}, & \mathrm{cent}(x) > t. \end{cases}$$

Equality is classified as *center* ($\le t$).

6.3 Strategy B: radial steps (`radial_steps`)

Parameters:

- an ordered list of metrics $(\mu_0, \ldots, \mu_{K-1})$ from center outward (default $(L1, L2, L\_\infty)$),
- a boundary scaling mode in {`linear`, `logarithmic`, `exponential`},
- a positive scaling parameter $b > 0$ (default $b = 2$).

The simulator precomputes $K - 1$ boundaries $0 < \beta_1 < \cdots < \beta_{K-1} < 1$.

Linear boundaries. With K metrics, define equally spaced internal cutpoints:

$$(\beta_1, \ldots, \beta_{K-1}) \coloneqq \left(\frac{1}{K}, \frac{2}{K}, \ldots, \frac{K-1}{K}\right).$$

Logarithmic boundaries. Let $\{\ell_i\}_{i=0}^{K}$ be $K + 1$ points logarithmically spaced between $b^0 = 1$ and $b^1 = b$, i.e. $\ell_i = b^{i/K}$. The code then normalizes these to $[0, 1]$ via:

$$\tilde{\ell}_i \coloneqq \frac{\ell_i - 1}{b - 1},$$

and uses the internal points $\beta_j = \tilde{\ell}_j$ for $j = 1, \ldots, K - 1$.

Exponential boundaries. Let $t_i = i/K$ for $i = 0, \ldots, K$. The code maps linearly spaced t_i through:

$$\tilde{e}_i \coloneqq \frac{b^{t_i} - 1}{b - 1},$$

and uses $\beta_j = \tilde{e}_j$ for $j = 1, \ldots, K - 1$.

Metric assignment. Given a voter position x, compute $c = \mathrm{cent}(x)$. The region index is:

$$r(c) \coloneqq \min\Big(\big|\{j \in \{1, \dots, K-1\} : \beta_j < c\}\big|,\ K-1\Big).$$

This matches the code's `searchsorted` behavior with the default left side: if $c = \beta_j$, then $r(c) = j - 1$, so boundaries are included in the *inner* region. Finally $\mu(x) \coloneqq \mu_{r(c)}$.

7 Utility formation

Given distances $D_{v,c}^{(p)}$, the simulator implements the following utility functions:

7.1 Gaussian utility (`gaussian`)

Let $\sigma_{\mathrm{fac}} > 0$ be a configuration parameter (default $\sigma_{\mathrm{fac}} = 0.5$). Define:

$$\sigma \coloneqq \sigma_{\mathrm{fac}} \sqrt{d}.$$

Utilities are:

$$u_{v,c}^{(p)} \coloneqq \exp\left(-\frac{(D_{v,c}^{(p)})^2}{2\sigma^2}\right).$$

7.2 Quadratic utility (`quadratic`)

Let $d_{\max} > 0$ be a configured maximum distance. If $d_{\max}$ is not provided, the code defaults to:

$$d_{\max} \coloneqq \sqrt{d},$$

which corresponds to the Euclidean diameter of the unit hypercube $[0, 1]^d$ and does *not* automatically adjust if geometry bounds are changed. Utilities are:

$$u_{v,c}^{(p)} \coloneqq \max\left(0,\ 1 - \left(\frac{D_{v,c}^{(p)}}{d_{\max}}\right)^2\right).$$

7.3 Linear utility with per-voter normalization (`linear`)

This is the most important case for heterogeneous distance studies because the normalization depends on each voter's (possibly heterogeneous) metric.

Opposite hypercube corner. For a voter position $x \in [p_{\min}, p_{\max}]^d$, define the "opposite corner" $o(x) \in \{p_{\min}, p_{\max}\}^d$ coordinate-wise:

$$o(x)_k \coloneqq \begin{cases} p_{\min}, & |x_k - p_{\min}| \geq |x_k - p_{\max}|, \\ p_{\max}, & |x_k - p_{\min}| < |x_k - p_{\max}|. \end{cases}$$

Ties ($=$) go to $p_{\min}$ (this is exactly the code's $\geq$ branch).

Per-voter maximum distance. Let μ_v be the metric assigned to voter v (heterogeneous) or the global metric (homogeneous). Define:

$$d_{\max,v} \coloneqq d_{\mu_v}\big(x_v,\, o(x_v)\big).$$

To avoid division by zero in degenerate cases, the code uses a "safe" denominator:

$$\tilde{d}_{\max,v} \coloneqq \begin{cases} d_{\max,v}, & d_{\max,v} > 10^{-12}, \\ 1, & \text{otherwise.} \end{cases}$$

Linear utility. Utilities are then:

$$u_{v,c}^{(p)} \coloneqq 1 - \frac{D_{v,c}^{(p)}}{\tilde{d}_{\max,v}},$$

and finally clipped to $[0, 1]$.

7.4 Non-implemented options

The configuration enum lists additional utility types (e.g. `exponential`, `saturated`), but the executed implementation raises an error for any utility function other than `gaussian`, `quadratic`, or `linear`. This chapter therefore treats only the implemented functions above.

8 From utilities to rankings

For each profile p and voter v, the simulator converts the utility vector $\big(u_{v,1}^{(p)}, \ldots, u_{v,M}^{(p)}\big)$ into an ordinal ranking by sorting in descending utility:

$$R_v^{(p)} \coloneqq \operatorname{argsort}\big(-u_{v,\cdot}^{(p)}\big).$$

No explicit ε-tie handling is applied; ties are broken by the underlying `argsort` ordering (which may be implementation-dependent).

9 Voting rules

Each profile is evaluated under one or more voting rules. Rules partition into:

- **Cardinal rules** consuming utilities $u_{v,c}$,
- **Ordinal rules** consuming rankings R_v.

9.1 Tie-breaking conventions

Some rules call a common tie-breaker on a vector of candidate scores $S \in \mathbb{R}^M$:

$$\operatorname{tiebreak}(S) \in \arg\max_c S_c.$$

If $|\arg\max S| = 1$, the unique maximizer is selected. Otherwise:

- `random`: pick uniformly at random among tied maxima using the simulator RNG,

- `lexicographic`: pick the smallest candidate index among ties,
- `none`: behaves identically to `lexicographic` in the current code.

Important: many rules use NumPy $\arg\max$ / $\arg\min$ internally without tiebreak; those ties resolve to the first index returned by NumPy.

9.2 Cardinal rules

Below, utilities are $u_{v,c}$ for $v = 1, \dots, N$, $c = 1, \dots, M$.

Utilitarian (`utilitarian`). Score each candidate by total utility:

$$S_c \coloneqq \sum_{v=1}^{N} u_{v,c}, \qquad \hat{c} \coloneqq \text{tiebreak}(S).$$

Approval (`approval`). Let $A_{v,c} \in \{0, 1\}$ indicate whether voter v approves candidate c. The code supports these approval policies:

- `top_k`: $A_{v,c} = 1$ iff c is among the k highest-utility candidates for voter v,
- `threshold`: $A_{v,c} = 1$ iff $u_{v,c} > \tau$,
- `mean`: $A_{v,c} = 1$ iff $u_{v,c} > \frac{1}{M} \sum_{j=1}^{M} u_{v,j}$,
- `above_average`: $A_{v,c} = 1$ iff $u_{v,c} > \frac{1}{NM} \sum_{v,j} u_{v,j}$.

Candidate scores and winner:

$$S_c \coloneqq \sum_{v=1}^{N} A_{v,c}, \qquad \hat{c} \coloneqq \text{tiebreak}(S).$$

Score / range voting (`score`). Let $L \in \mathbb{N}$ be the configured maximum score (`score_max`, default $L = 5$). Each voter rescales their own utilities independently:

$$u_v^{\min} \coloneqq \min_c u_{v,c}, \qquad u_v^{\max} \coloneqq \max_c u_{v,c}, \qquad \rho_v \coloneqq u_v^{\max} - u_v^{\min},$$

with ρ_v replaced by 1 if $\rho_v = 0$. Normalized utilities:

$$\tilde{u}_{v,c} \coloneqq \frac{u_{v,c} - u_v^{\min}}{\rho_v} \in [0, 1].$$

Ballots:

$$b_{v,c} \coloneqq \begin{cases} \text{round}(L\, \tilde{u}_{v,c}), & \text{if granularity is integer,} \\ L\, \tilde{u}_{v,c}, & \text{if granularity is continuous.} \end{cases}$$

Scores and winner:

$$S_c \coloneqq \sum_{v=1}^{N} b_{v,c}, \qquad \hat{c} \coloneqq \text{tiebreak}(S).$$

STAR (`star`). First compute score totals S_c exactly as in `score`. Let the finalists be the two candidates with highest totals:

$$(a, b) \coloneqq \text{the first two indices in argsort}(-S).$$

Then perform an automatic runoff using the *original utilities* (not the rounded score ballots):

$$V_a \coloneqq \left|\{v : u_{v,a} > u_{v,b}\}\right|, \qquad V_b \coloneqq \left|\{v : u_{v,b} > u_{v,a}\}\right|.$$

If $V_a > V_b$ select a; if $V_b > V_a$ select b. If $V_a = V_b$, select the finalist with greater (or equal) score total S (i.e. choose a if $S_a \geq S_b$, else choose b).

Median utility (`median`). Compute per-candidate medians:

$$S_c \coloneqq \text{median}\big(u_{1,c}, \ldots, u_{N,c}\big), \qquad \hat{c} \coloneqq \text{tiebreak}(S).$$

Quadratic voting (`quadratic`). The code applies a signed square root transform to utilities:

$$w_{v,c} \coloneqq \text{sign}(u_{v,c}) \sqrt{|u_{v,c}|}, \qquad S_c \coloneqq \sum_{v=1}^{N} w_{v,c}, \qquad \hat{c} \coloneqq \text{tiebreak}(S).$$

Note: when utilities are nonnegative (as in this simulator's implemented utility functions), this reduces to $w_{v,c} = \sqrt{u_{v,c}}$.

9.3 Ordinal rules

Below, R_v is voter v's ranking permutation, with $R_v[0]$ the top choice.

Plurality (`plurality`). First-choice counts:

$$S_c \coloneqq \left|\{v : R_v[0] = c\}\right|, \qquad \hat{c} \coloneqq \text{tiebreak}(S).$$

Anti-plurality / veto (`anti_plurality`, `veto`). Last-place counts $L_c \coloneqq |\{v : R_v[M-1] = c\}|$. The code sets scores $S_c = -L_c$ and applies tiebreak.

Borda (`borda`). Assign points $M-1, M-2, \ldots, 0$ to ranks $0, 1, \ldots, M-1$. Score:

$$S_c \coloneqq \sum_{v=1}^{N} \big(M - 1 - \text{rank}_v(c)\big),$$

where $\text{rank}_v(c)$ is the index r such that $R_v[r] = c$. Winner: $\hat{c} = \text{tiebreak}(S)$.

IRV (`irv`). Maintain an active set $A \subseteq \{0, \ldots, M-1\}$, initially all candidates active. In each round:

- each voter contributes one vote to their highest-ranked active candidate,
- if some candidate has strictly more than half of active votes, that candidate wins,
- otherwise eliminate the active candidate with the *fewest* first-choice votes (ties resolve by smallest index via arg min behavior), remove it from A, and repeat.

If no strict majority occurs, the last remaining active candidate wins.

Coombs (`coombs`). As in IRV, but elimination removes the active candidate with the *most* last-place votes among active candidates (ties resolve by smallest index via arg max).

Condorcet winner test (`condorcet`). Compute pairwise margins (Section 11.5). A Condorcet winner is a candidate that beats every other candidate by positive margin. If none exists, the voting-rule engine returns winner index -1.

Minimax (`minimax`). Compute pairwise margins $m_{i,j}$. For each candidate c, define its worst defeat as:

$$W_c \coloneqq \max_{j \neq c}(-m_{c,j}),$$

and set score $S_c \coloneqq -W_c$. Winner is tiebreak(S).

Copeland (`copeland`). Let $w_c = |\{j \neq c : m_{c,j} > 0\}|$ and $\ell_c = |\{j \neq c : m_{c,j} < 0\}|$. Score $S_c \coloneqq w_c - \ell_c$. Winner is tiebreak$(S)$.

Schulze / beatpath (`schulze`). Compute pairwise margins and initialize a "strength" matrix:

$$P_{i,j} \coloneqq \max(m_{i,j}, 0) \quad (i \neq j), \quad P_{i,i} = 0.$$

Then apply the Floyd–Warshall-style update:

$$P_{i,j} \leftarrow \max\big(P_{i,j},\ \min(P_{i,k}, P_{k,j})\big) \quad \text{for all } k, i, j,\ i \neq j.$$

Finally, the code counts for each i how many $j \neq i$ satisfy $P_{i,j} > P_{j,i}$, and selects the maximum by tiebreak.

Ranked pairs / Tideman (`ranked_pairs`). Build the set of directed pair victories (w, ℓ) with a positive margin, each annotated by its margin value. Sort these victories by decreasing margin. Iteratively "lock" a victory unless it would create a directed cycle in the locked graph. The winner is the candidate with no incoming edge in the locked graph; if multiple, the code selects the smallest index by scanning from 0 upward.

Bucklin (`bucklin`). Initialize scores $S_c = 0$. For $r = 0, 1, \ldots, M-1$:

- add 1 to each voter's candidate at rank r,
- if $\max_c S_c$ exceeds $N/2$ (strictly), select arg max S (ties by smallest index).

If no majority ever occurs, select tiebreak(S) after the final round.

Nanson (`nanson`). Iteratively compute Borda scores on the active set and eliminate all candidates with below-average Borda score among active candidates. If no candidate is eliminated in a round (all remaining have score $\geq$ average), stop. Winner is the smallest index remaining active.

Baldwin (`baldwin`). Iteratively compute Borda scores on the active set and eliminate the single candidate with lowest Borda score (ties by smallest index). Winner is the last remaining active candidate (smallest index if a tie in active representation).

Kemeny–Young (`kemeny_young`). If $M > 8$, the code falls back to the Schulze method. Otherwise it enumerates all permutations π of candidates and scores each permutation by the sum of pairwise margins consistent with π:

$$K(\pi) \coloneqq \sum_{i<j} m_{\pi_i,\pi_j}.$$

It selects a maximizer $\pi^\star$ (first encountered in enumeration order among ties) and declares $\pi_0^\star$ the winner.

10 Strategic manipulation (optional)

Manipulation is optional. When enabled, the simulator selects a subset of voters as manipulators and then modifies either:

- utilities (for `bullet` strategy under cardinal rules), and/or
- rankings (for ordinal-rule strategies `compromise`, `burial`, `pushover`, `optimal`).

When a rule is cardinal, the simulator passes manipulated *utilities* into the voting rule; when a rule is ordinal, it passes manipulated *rankings*. Thus, in the implemented code, most ranking-only manipulation strategies do not affect cardinal rules.

10.1 Manipulator fraction and truncation

Let $f \in [0, 1]$ be the configured manipulator fraction. The number of manipulators is:

$$N_{\text{manip}} \coloneqq \lfloor fN \rfloor,$$

clipped to $[0, N]$.

10.2 Manipulator selection methods

The code supports:

- `random`: uniform subset of size N_{manip} without replacement,
- `extremists`: pick voters with largest variance $\text{Var}_c(u_{v,c})$,
- `centrists`: pick voters minimizing $\sum_c (u_{v,c} - \bar{u}_c)^2$ where $\bar{u}_c = \frac{1}{N}\sum_v u_{v,c}$,
- `informed`: approximate expected winner by plurality on argsort$(-u)$ first choices; pick voters maximizing $\max_c u_{v,c} - u_{v,c^\star}$.

10.3 Poll information (for `polls` and `full` information levels)

The polling model uses first-choice counts C_c from sincere rankings.

- `full`: vote shares are C_c/N,
- `polls`: add Gaussian noise $C_c + \eta_c$ with $\eta_c \sim \mathcal{N}(0, \sigma^2)$, $\sigma = \texttt{poll\_noise} \cdot N$; clamp at 0 and renormalize to shares.

The set of "viable" candidates is those with share > 0.1 (in `polls`) or count $> 0.1N$ (in `full`).

10.4 Manipulation strategies

Bullet (`bullet`). Only applied for cardinal rules. For each manipulating voter v, let $c_1 = R_v[0]$ be their sincere top choice. Set:

$$u'_{v,c} \coloneqq 0 \;\; (c \neq c_1), \qquad u'_{v,c_1} \coloneqq u_{v,c_1}.$$

Compromise (`compromise`). Only modifies rankings and only when poll info is available. If a manipulator's top choice is not viable, the code promotes the first viable candidate appearing in their ranking to first place and keeps the remaining candidates in their original relative order.

Burial (`burial`). Only modifies rankings and only when poll info is available. If the poll "frontrunner" is not the voter's top choice, move the frontrunner to last place.

Pushover (`pushover`). Only modifies rankings and only when poll info is available. Let the "weakest" candidate be the one with smallest poll vote share. If that candidate is not already first, the code inserts the weakest candidate into *second* position (index 1) while keeping the voter's sincere top choice in first place.

Optimal (`optimal`). For $M \leq 5$, for each manipulator v, the code enumerates all possible rankings π and selects the one that maximizes the manipulator's *sincere* utility for the resulting winner under the chosen voting rule. Note that for *cardinal* rules, the implementation only varies rankings while passing unchanged utilities into the rule, so the winner is unaffected by π and the "optimal" search is effectively vacuous. For $M > 5$, the code falls back to a compromise call without poll info (which returns without changing rankings, i.e. performs no change).

11 Evaluation metrics

Metrics are computed *using sincere utilities and rankings*, even if manipulation is enabled, so that welfare comparisons are measured against the true underlying preferences.

11.1 Social utility vector

Define per-candidate mean utility:

$$\bar{u}_c \coloneqq \frac{1}{N} \sum_{v=1}^{N} u_{v,c}.$$

For a selected winner $\hat{c}$, define:

$$U_{\text{win}} \coloneqq \bar{u}_{\hat{c}}, \quad U_{\max} \coloneqq \max_c \bar{u}_c, \quad U_{\min} \coloneqq \min_c \bar{u}_c.$$

11.2 Voter Satisfaction Efficiency (VSE)

Let $\Delta U = U_{\max} - U_{\min}$. If $\Delta U > \varepsilon$, the code computes:

$$\text{VSE} \coloneqq \frac{U_{\text{win}} - U_{\min}}{\Delta U}.$$

Otherwise (all candidates essentially tied), it sets VSE $= 1$.

11.3 Regret

Regret is:

$$\text{Regret} \coloneqq U_{\max} - U_{\text{win}},$$

and a normalized regret ratio Regret/ΔU is computed when $\Delta U > \varepsilon$, else 0.

11.4 Winner rank by social utility

Let π be candidates sorted by descending $\bar{u}_c$ (via argsort$(-\bar{u})$). The winner rank is the index r such that $\pi_r = \hat{c}$. Ties follow argsort ordering.

11.5 Pairwise margins and Condorcet winner

From rankings R_v, define pairwise margin matrix $m \in \mathbb{Z}^{M \times M}$:

$$m_{i,j} \coloneqq |\{v : i \text{ is ranked above } j \text{ by } v\}| - |\{v : j \text{ is ranked above } i \text{ by } v\}|.$$

A Condorcet winner exists if some c satisfies $m_{c,j} > 0$ for all $j \neq c$.

11.6 Cycle detection and classification

For $M = 3$, the code declares a cycle iff no Condorcet winner exists. It classifies cycles as:

- type 1 if $0 \succ 1$, $1 \succ 2$, and $2 \succ 0$ by positive margins,
- type 2 if $1 \succ 0$, $0 \succ 2$, and $2 \succ 1$ by positive margins.

For $M > 3$, the code constructs the directed graph of strict pairwise-majority edges $\{i \to j : m_{i,j} > 0\}$ and applies a DFS-like check for cyclic reachability (as implemented).

12 Pipeline summary

For each profile $p = 1, \ldots, P$:

1. sample voter and candidate positions $\{x_v^{(p)}\}, \{y_c^{(p)}\}$ according to the chosen geometry (Section 4);
2. compute distances $D_{v,c}^{(p)}$ either with one metric or with heterogeneous per-voter metrics (Sections 5–6);
3. compute utilities $u_{v,c}^{(p)}$ from distances (Section 7);
4. compute rankings $R_v^{(p)}$ by sorting utilities (Section 8);
5. optionally apply manipulation to obtain manipulated ballots (Section 10);
6. apply a voting rule to select a winner $\hat{c}^{(p)}$ (Section 9);
7. if the chosen rule returns "no winner" (only `condorcet` does so), the simulator falls back to the utilitarian winner $\arg\max_c \bar{u}_c$;
8. compute metrics (VSE, regret, Condorcet/cycle properties) using sincere utilities and rankings (Section 11).

13 Research methodology implemented in the repository

The repository contains two research harnesses (`heterogenity-simulator/` and `heterogeneity-research/`) that instantiate the simulator repeatedly to measure the effect of heterogeneous distance on outcomes. This section documents the *executed methodology* of the main comprehensive suite (`heterogenity-simulato`) it is the procedure used to generate the large JSON result files in `heterogenity-simulator/results/`.

13.1 Common experimental configuration

Unless explicitly varied by a phase, the research suite uses:

- geometry: `uniform` in dimension d with bounds $p_{\min} = -1$, $p_{\max} = 1$,
- utility: `linear` (Section 7.3),
- voting rules: $\{\texttt{plurality}, \texttt{borda}, \texttt{irv}\}$,
- number of candidates: $M = 5$,
- number of profiles: $P = 200$,
- random seed: $s = 42$.

13.2 Heterogeneous vs. homogeneous baselines

The suite focuses on the `center_extreme` heterogeneity strategy (Section 6.2). For a specified ordered metric pair $(\mu_{\text{center}}, \mu_{\text{extreme}})$ and threshold t, it runs:

- a **heterogeneous** configuration with $\mu(x) = \mu_{\text{center}}$ for $\text{cent}(x) \le t$ and $\mu(x) = \mu_{\text{extreme}}$ otherwise,
- a **homogeneous center baseline** using the single metric μ_{center},
- a **homogeneous extreme baseline** using the single metric μ_{extreme}.

For each voting rule, the suite records:

- **VSE** and **cycle / Condorcet metrics** (Section 11) for heterogeneous and baseline runs,
- **VSE differences** (heterogeneous minus baseline),
- **disagreement rate** between heterogeneous and baseline winners:

$$\text{Disagree} \coloneqq 100 \cdot \frac{1}{P} \sum_{p=1}^{P} \mathbf{1}\left[\hat{c}_{\text{het}}^{(p)} \neq \hat{c}_{\text{homo}}^{(p)}\right].$$

13.3 Phases of the comprehensive suite

The suite executes the following phases:

- **Voter scaling**: vary $N \in \{10, 25, 50, 100, 200, 300, 400, 500\}$ with fixed $d = 2$, $t = 0.5$, and default metrics $(L2, \cos)$.
- **Threshold sweep**: vary t over 19 values $t \in \{0.05, 0.10, \ldots, 0.95\}$ with fixed $N = 100$, $d = 2$, metrics $(L2, \cos)$.

- **Dimensional scaling**: vary $d \in \{1, 2, 3, 4, 5, 7, 10\}$ with fixed $N = 100$, $t = 0.5$, metrics $(L2, \cos)$.
- **Metric-pair interactions**: for the metric set $\{L1, L2, \cos, L_\infty\}$, run all ordered pairs $(\mu_{\text{center}}, \mu_{\text{extreme}})$ with $\mu_{\text{center}} \neq \mu_{\text{extreme}}$ at fixed $N = 100$, $d = 2$, $t = 0.5$. The suite also runs the reversed direction $(\mu_{\text{extreme}}, \mu_{\text{center}})$ and reports the absolute difference in disagreement rates as an *asymmetry* measure.
- **Final verification**: repeats the metric-pair interaction experiment at $N = 500$ (same d, t, P) to assess stability at larger electorates.