Decision Modeling

VOTING RULES AND COLLABORATIVE FILTERING

Professor: Brice Mayag

Students:
Dilbar Isakova
MD Kamrul Islam

November 18th, 2024

CentraleSupélec

VOTING RULES: THEORY

The goal of the problem: **output should provide the results of an election simulation where *4 voting methods* were used, and all of them produced the same winner.**

- ***Plurality Voting:*** **each voter selects one candidate. The candidate with the most votes wins.**
- ***Plurality with Runoff:*** **if no candidate gets more than 50% of the votes, the top two candidates go to a second round. In the runoff, the candidate with the most votes wins.**
- ***Condorcet:*** **the candidate who would beat every other candidate in one-on-one matchups is the winner.**
- ***Borda:*** **voters rank candidates. Points are assigned based on rankings:**

  ***1st place = more points.**

  ***2nd place = fewer points, and so on.**

  *** the candidate with the most total points wins.**

VOTING RULES: CONTRAINTS

- **At least 20% of voters should have distinct preferences.**
- **No single candidate should dominate as the best candidate for more than 70% of voters.**

```
def check_minimum_different_preferences(preferences, min_percentage=0.20):
    unique_preferences = set(preferences)
    return len(unique_preferences) >= len(preferences) * min_percentage

def check_max_same_best_candidate(preferences, max_percentage=0.70):
    first_choices = [p[0] for p in preferences]
    first_choice_counts = Counter(first_choices)
    max_first_choice_count = max(first_choice_counts.values())
    return max_first_choice_count <= len(preferences) * max_percentage
```

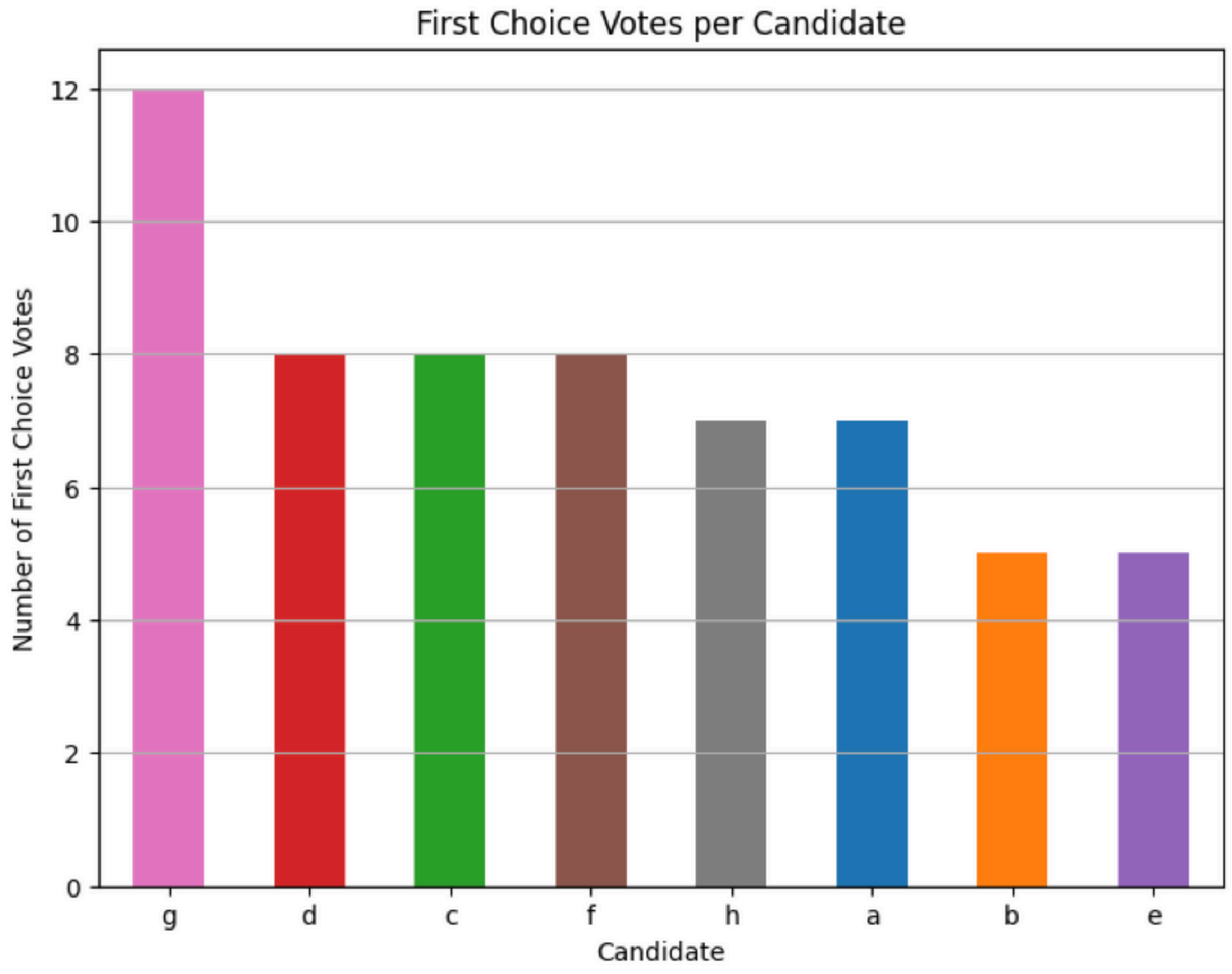
VOTING RULES: METHOD

```
for attempt in range(10000):
    preferences = generate_voter_preferences(n_voters, candidates)

    if check_minimum_different_preferences(preferences) and check_max_same_best_ca
        results, is_consistent = validate_data(preferences, candidates)

        if is_consistent and results['Condorcet'][0] is not None:
            saved_preferences = preferences
            ...
            break
else:
    print("Could not find a suitable election within 10,000 attempts.")
```

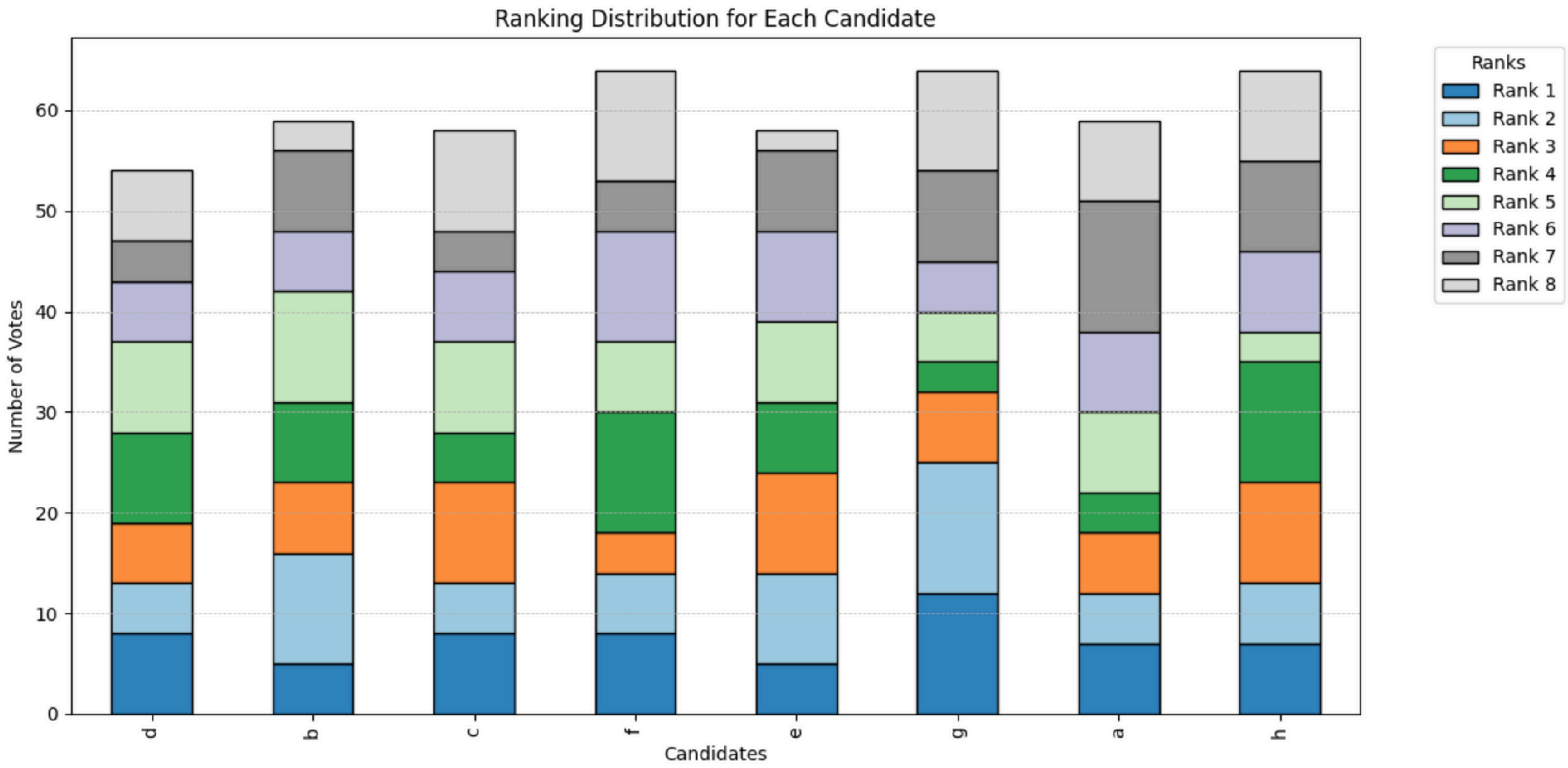
- **Randomly generating voter preferences repeatedly (up to 10,000 times).**
- **Checking each set of preferences to see if they meet the conditions.**
- **Stopping only when a suitable set is found or when all attempts fail.**

VOTING RULES: QUESTION 5

```
Election Results (Consistent winner found on attempt 4):

1. Plurality Voting
   Winner: g
2. Plurality Runoff Voting
   Winner: g
3. Condorcet Voting
   Winner: g

4. Borda Voting
   Winner: g

Saved Preferences (Voter Rankings):
   Rank 1 Rank 2 Rank 3 Rank 4 Rank 5 Rank 6 Rank 7 Rank 8
0       g      a      f      d      b      e      h      c
1       f      c      b      a      e      d      g      h
2       d      f      e      c      b      h      a      g
3       h      d      a      g      f      e      c      b
4       d      g      e      a      b      f      c      h
5       d      h      f      b      e      g      a      c
6       f      g      d      c      h      e      b      a
7       h      c      f      a      d      g      b      e
8       e      b      g      a      f      c      d      h
9       c      g      b      d      e      h      f      a
10      e      a      b      d      g      f      h      c
11      h      g      d      c      f      e      h      a
```

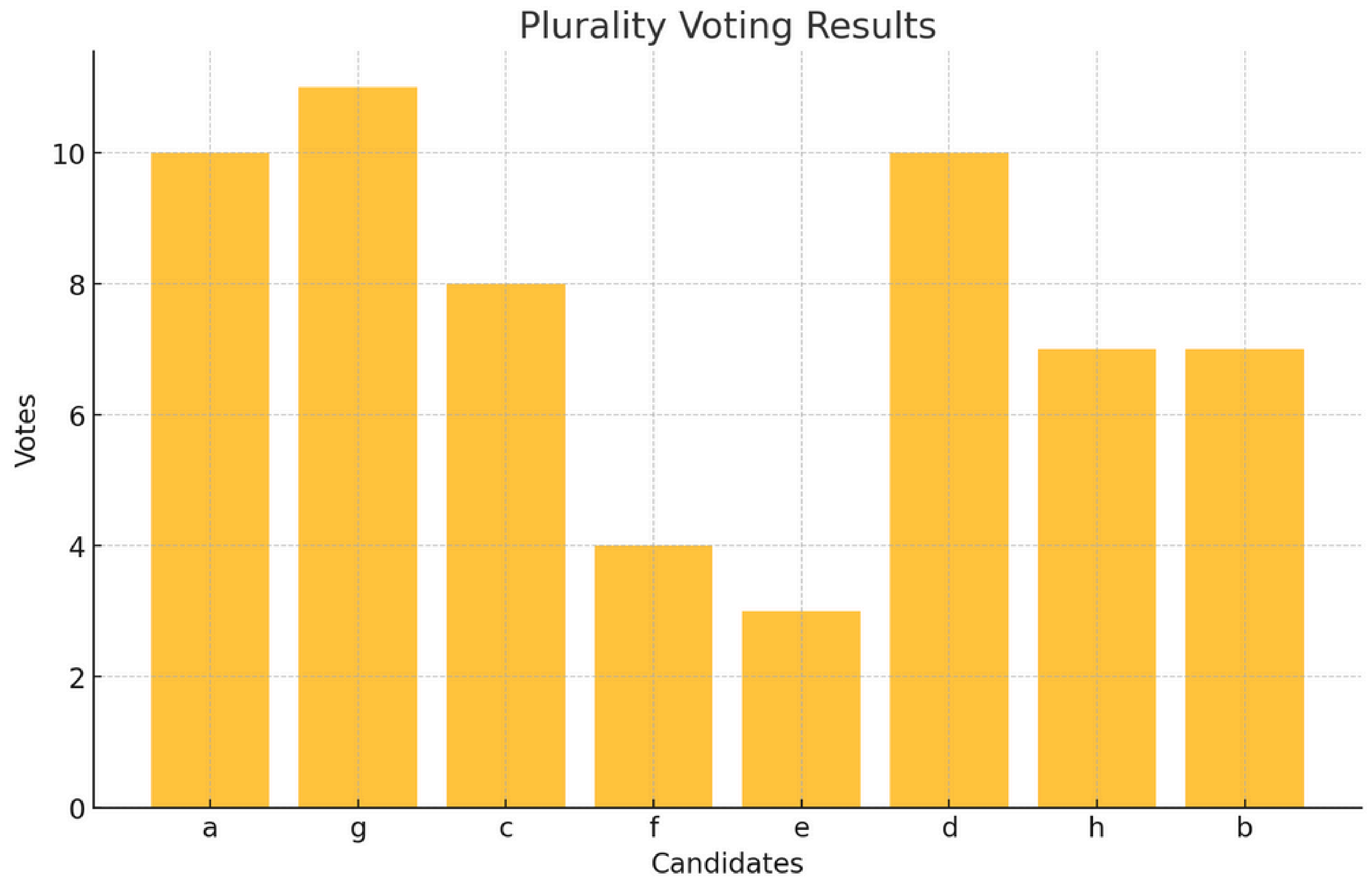


VOTING RULES: OUR CASE



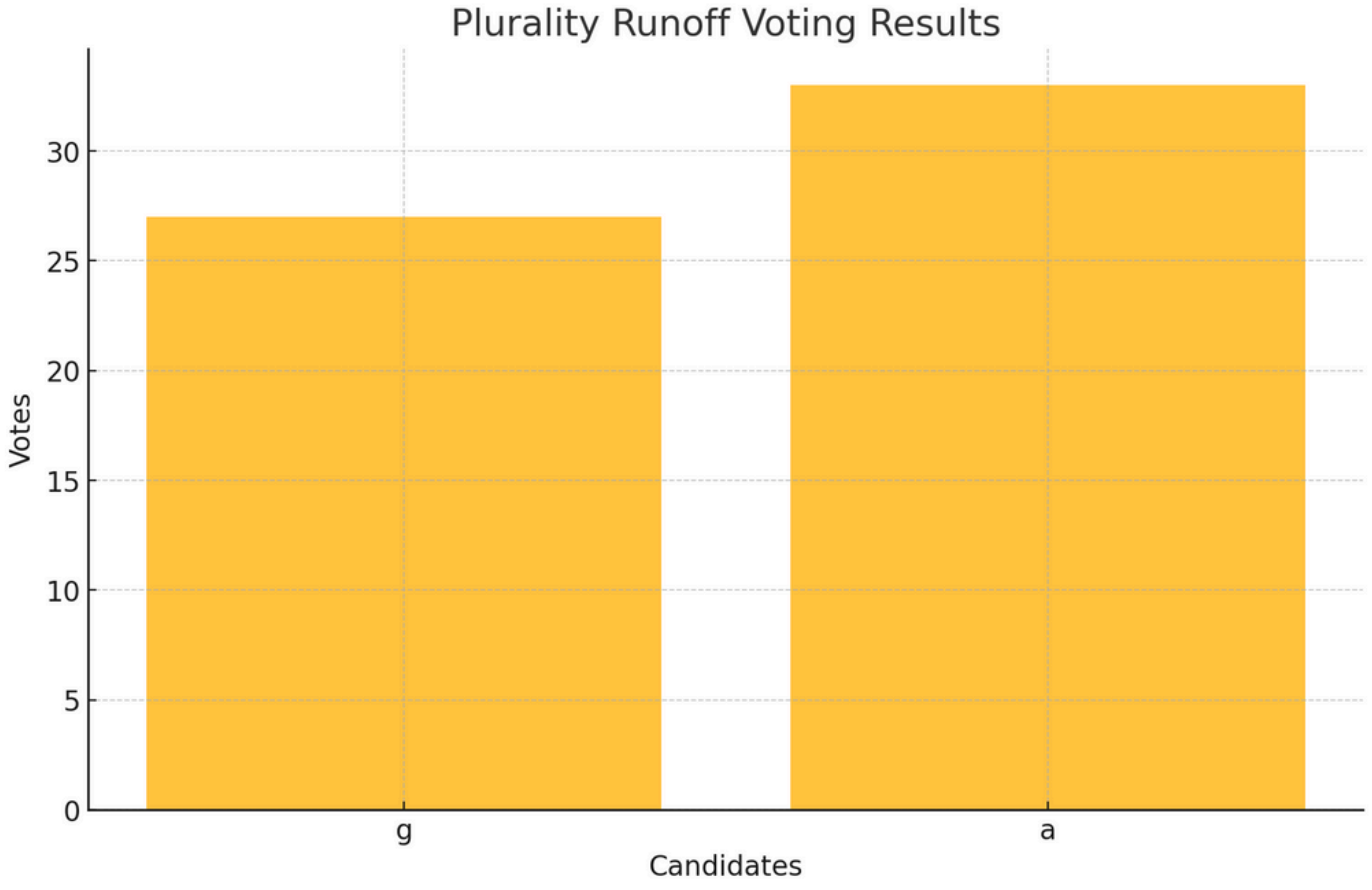
CentraleSupélec

VOTING RULES: QUESTION 6




```
Unique Election Results (Found on attempt 4):

Plurality Winner: g
Plurality Runoff Winner: a
Condorcet Winner: None
Borda Winner: h

Voter Preferences (Sample):
   Rank 1 Rank 2 Rank 3 Rank 4 Rank 5 Rank 6 Rank 7 Rank 8
0       a      d      f      e      g      h      c      b
1       g      e      h      c      d      b      f      a
2       c      f      g      h      d      e      a      b
3       a      f      b      c      e      d      g      h
4       c      h      f      d      b      a      g      e
5       a      g      e      d      b      f      c      h
6       g      h      f      a      c      d      b      e
7       f      a      g      b      h      c      e      d
8       c      h      b      d      f      e      a      g
9       g      c      h      e      a      d      f      b
10      g      e      h      c      d      f      a      b
11      e      g      d      f      a      h      h      c
```

VOTING RULES: QUESTION 6



Unique Election Results (Found on attempt 4):

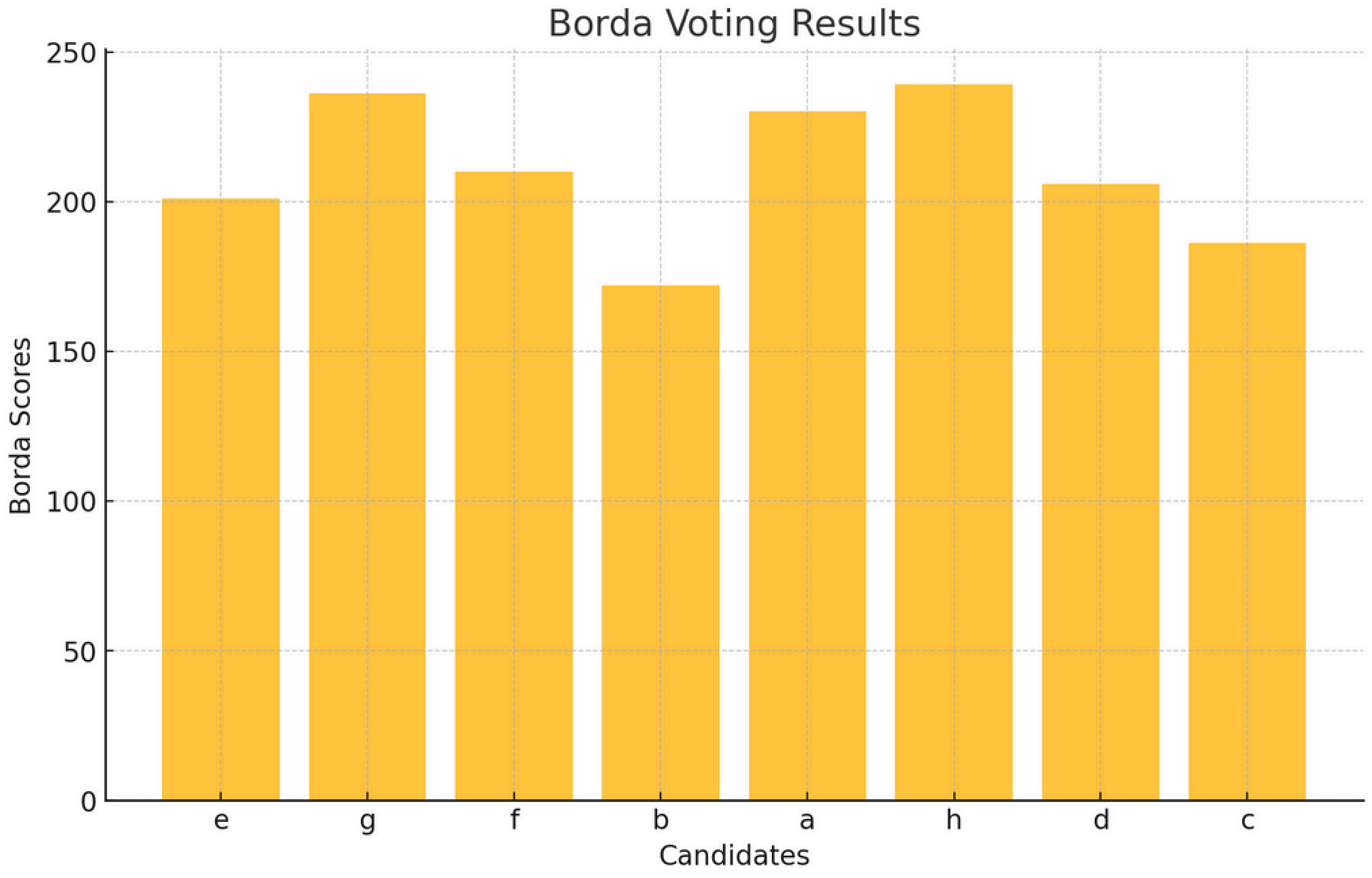
Plurality Winner: g
Plurality Runoff Winner: a
Condorcet Winner: None
Borda Winner: h

Voter Preferences (Sample):

	Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Rank 8
0	a	d	f	e	g	h	c	b
1	g	e	h	c	d	b	f	a
2	c	f	g	h	d	e	a	b
3	a	f	b	c	e	d	g	h
4	c	h	f	d	b	a	g	e
5	a	g	e	d	b	f	c	h
6	g	h	f	a	c	d	b	e
7	f	a	g	b	h	c	e	d
8	c	h	b	d	f	e	a	g
9	g	c	h	e	a	d	f	b
10	g	e	h	c	d	f	a	b
11	e	g	d	f	a	h	h	c

CentraleSupélec

VOTING RULES: QUESTION 6




```
Unique Election Results (Found on attempt 4):

Plurality Winner: g
Plurality Runoff Winner: a
Condorcet Winner: None
Borda Winner: h

Voter Preferences (Sample):
   Rank 1 Rank 2 Rank 3 Rank 4 Rank 5 Rank 6 Rank 7 Rank 8
0       a      d      f      e      g      h      c      b
1       g      e      h      c      d      b      f      a
2       c      f      g      h      d      e      a      b
3       a      f      b      c      e      d      g      h
4       c      h      f      d      b      a      g      e
5       a      g      e      d      b      f      c      h
6       g      h      f      a      c      d      b      e
7       f      a      g      b      h      c      e      d
8       c      h      b      d      f      e      a      g
9       g      c      h      e      a      d      f      b
10      g      e      h      c      d      f      a      b
11      e      g      d      f      a      h      b      c
```

PERFORMANCE ANALYSIS: VOTERS

Plurality and Plurality Runoff:

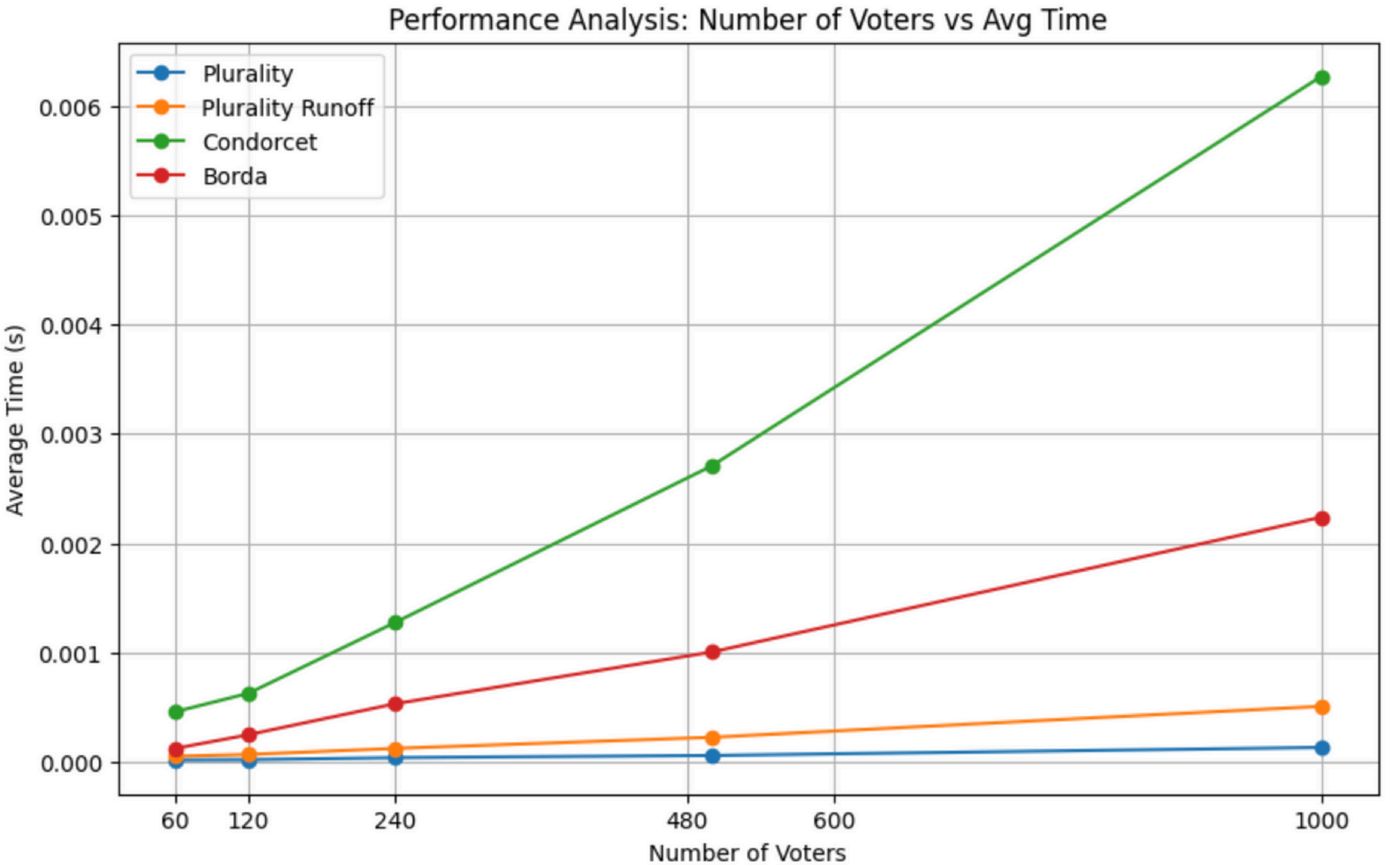
- **the most efficient methods**
- **computation times almost constant**
- **N of voters increases but still good for large-scale elections.**

Borda:

- **has moderate computational cost:**
- **scales linearly with the N of voter**
- **Plurality(Runoff)<Borda<Condorcet**

Condorcet:

- **most computationally expensive method**



COLLABORATIVE FILTERING
CONSISTENT RECOMMENDATIONS

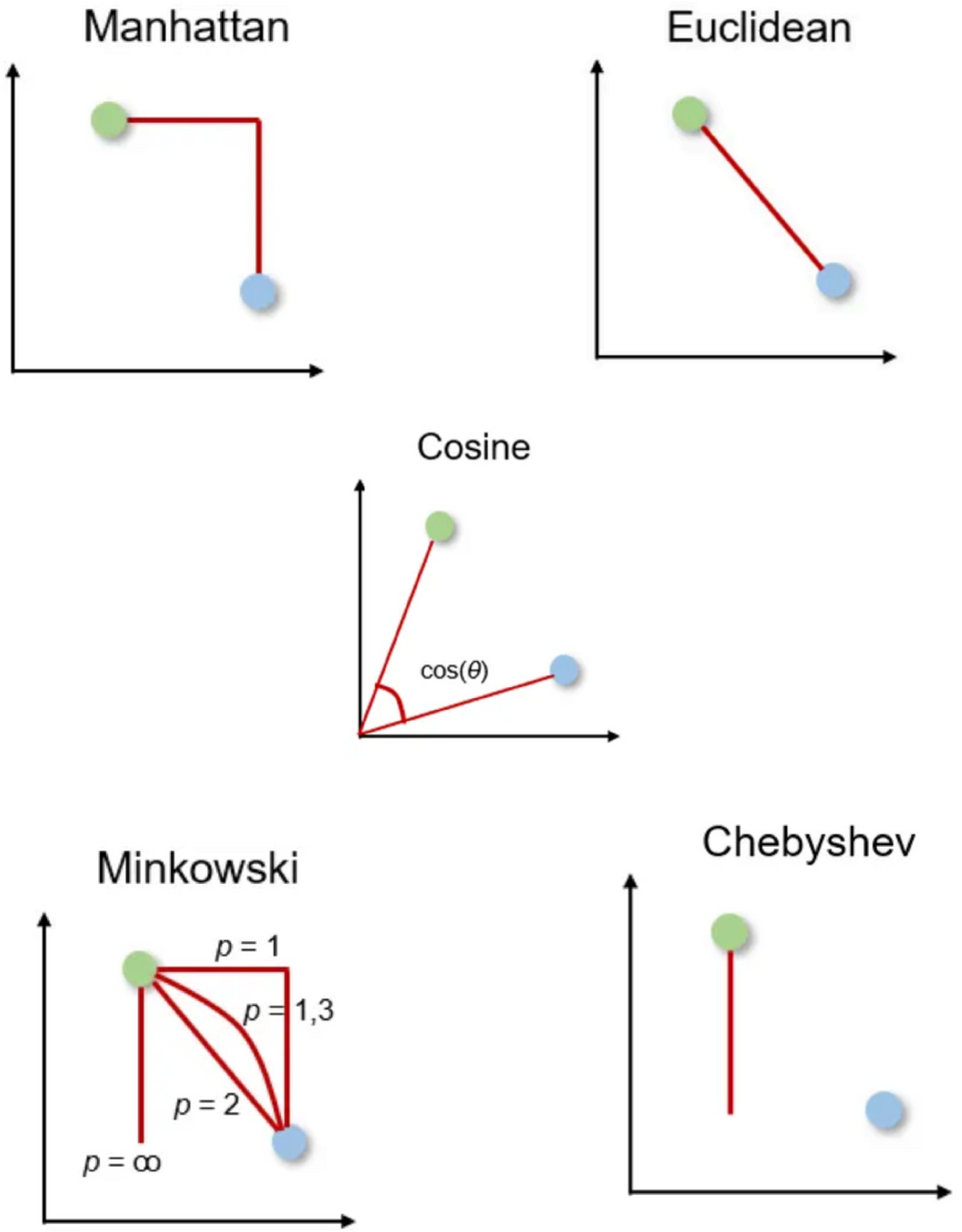
Objective: Ensure all six similarity measures recommend the same movie for a critic.

Dataset:

- Balanced Ratings
- One of the critic who has not seen at least half of the movies
- Number of critics ($n \geq 15$) and movies ($m \geq 20$).
- Missing data between 30%-50%.

SIMILARITY MEASURES

- Manhattan Distance (L1 norm): maximum difference in ratings
- Euclidean Distance (L2 norm):
- Minkowski Distance (generalized norm, p=3)
- Chebyshev Distance (maximum difference in ratings)
- Pearson Correlation (measuring linear relationship)
- Cosine Similarity (measuring the cosine of the angle between vectors)



METHODS FOR CONSISTENT RECOMMENDATIONS

Input: ratings_matrix, target_critic, movies_to_consider, similarity_measures

output: A ranked list of recommended movies for the target critic using each similarity measure.

Steps:

1. Data Generation
2. Define Similarity Functions
3. Find Shared Movies
4. Aggregate Weighted Scores
5. Rank Movies
6. Generate Recommendations

$$\text{Weighted Score} = \sum \left(\frac{\text{Rating} \times \text{Weight}}{\text{Sum of Weights}} \right)$$

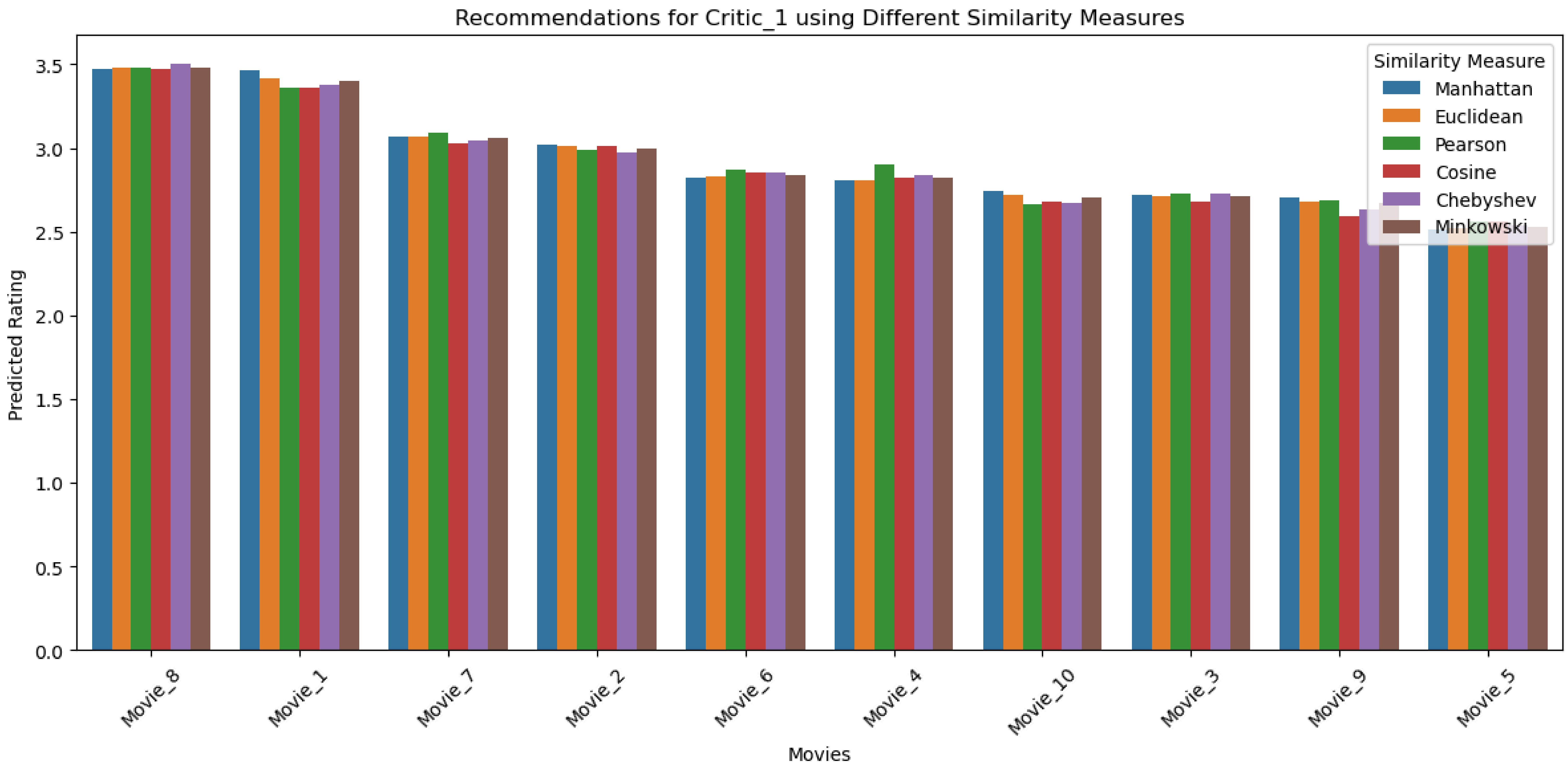
Weight is defined as $w = \frac{1}{1+\text{distance}}$.

RESULT

Top Recommendations for Each Similarity Measure

Similarity Measure	Rank	Movie	Score
Chebyshev	1	Movie_8	3.5
Chebyshev	2	Movie_1	3.38
Chebyshev	3	Movie_7	3.04
Cosine	1	Movie_8	3.47
Cosine	2	Movie_1	3.36
Cosine	3	Movie_7	3.03
Euclidean	1	Movie_8	3.48
Euclidean	2	Movie_1	3.42
Euclidean	3	Movie_7	3.07
Manhattan	1	Movie_8	3.47
Manhattan	2	Movie_1	3.46
Manhattan	3	Movie_7	3.07
Minkowski	1	Movie_8	3.48
Minkowski	2	Movie_1	3.4
Minkowski	3	Movie_7	3.06
Pearson	1	Movie_8	3.48
Pearson	2	Movie_1	3.36
Pearson	3	Movie_7	3.09

RESULT



- **All six similarity measures produced nearly identical recommendations for the target critic.**
- **"Movie_8" and "Movie_1" consistently ranked as the highest recommendations across all measures.**

COLLABORATIVE FILTERING

DIFFERENT RECOMMENDATIONS

Objective: Ensure all six similarity measures recommend different movies for a critic.

Dataset:

- Conflicting Ratings (variability in ratings)
- At least half the movies are unseen by the targeted critic
- Number of critics ($n \geq 15$) and movies ($m \geq 20$).
- Missing data between 30%-50%.
- Increased variablity, for example,

Critics 1–25: Strongly favor specific movies and dislike others.

Critics 26–50: The opposite preference pattern.

METHOD FOR DIFFERENT RECOMMENDATIONS

1. Simulating Data (50 critics rated 250 movies, and rated less than half the movies)
2. Similarity Measures
3. Weighted Global Scoring for Recommendations

$$s'(a) = \frac{\sum_{x \in C(a)} \mathrm{Weight}(x, \mathrm{target}) \cdot x(a)}{\sum_{x \in C(a)} \mathrm{Weight}(x, \mathrm{target})}$$

where, Weight(x,target)=exp(−similarity)

4. Iterative Recommendation Process : After each iteration, movie recommendations were recalculated based on updated similarity weights, Iterations continued until the recommendations stabilized
5. Final Recommendations

RESULT

```
Recommendations have converged.
Final Recommendations using Pearson similarity:
1. Movie_26 (Score: 3.53)
2. Movie_116 (Score: 3.50)
3. Movie_31 (Score: 3.49)
4. Movie_62 (Score: 3.48)
5. Movie_245 (Score: 3.48)

Final Recommendations using Cosine similarity:
1. Movie_26 (Score: 3.54)
2. Movie_116 (Score: 3.54)
3. Movie_62 (Score: 3.51)
4. Movie_31 (Score: 3.48)
5. Movie_75 (Score: 3.45)

Final Recommendations using Manhattan similarity:
1. Movie_18 (Score: 5.00)
2. Movie_169 (Score: 5.00)
3. Movie_207 (Score: 5.00)
4. Movie_92 (Score: 5.00)
5. Movie_91 (Score: 5.00)

Final Recommendations using Euclidean similarity:
1. Movie_116 (Score: 4.09)
2. Movie_69 (Score: 3.80)
3. Movie_18 (Score: 3.80)
4. Movie_91 (Score: 3.74)
5. Movie_169 (Score: 3.72)

Final Recommendations using Chebyshev similarity:
1. Movie_26 (Score: 3.63)
2. Movie_75 (Score: 3.56)
3. Movie_88 (Score: 3.56)
4. Movie_116 (Score: 3.53)
5. Movie_69 (Score: 3.51)

Final Recommendations using Minkowski similarity:
1. Movie_116 (Score: 3.79)
2. Movie_31 (Score: 3.58)
3. Movie_69 (Score: 3.57)
4. Movie_24 (Score: 3.52)
5. Movie_198 (Score: 3.49)
```

THANK YOU!