Journal of Advances in Mathematics and Computer Science

xx(x): ....., 20yy; Article no.JAMCS.xxxxx

ISSN: 2456-9968

(Past name: British Journal of Mathematics & Computer Science, Past ISSN: 2231-0851)

Implementation of the VDEPCI Method for Large-Scale Elections

Original Research Article

Received: XX December 20XX
Accepted: XX December 20XX
Online Ready: XX December 20XX

Abstract

Across the world, and particularly in Africa, elections are increasingly marked by protest movements, often resulting from a sense of injustice or from the contestation of results. It therefore becomes essential to adopt a voting system that promotes collective acceptance and reflects a general consensus. Indeed, it is not uncommon for certain candidates to refuse to acknowledge their defeat, thereby fueling post-electoral tensions. In this context, the VDEPCI method (Vote based on a Distance for Evaluating Preferences with respect to the Ideal Candidate) has been proposed as a promising alternative. It aims to minimize divergences by taking into account both voters' preferences and a confidence index in the rankings. However, applications of this method have so far been limited to simple cases, involving a small number of candidates and voters. Our research work is part of a perspective of extending this method to more complex electoral contexts, with a larger number of candidates and voters (several thousands), by emphasizing algorithmic implementation and the optimization of the computations required for its application. The main computational result establishes that the proposed implementation has a linear time complexity, $\mathcal{O}(n \times m)$, with respect to the number of voters and candidates. Experimental results confirm excellent large-scale performance, demonstrating that the method can efficiently handle elections involving several thousands of voters while maintaining very low execution times.

Keywords: VDEPCI method, ideal candidate, algorithmic implementation, election, consensus
2010 Mathematics Subject Classification: 53C25; 83C05; 57N16

1 Introduction

Social choice theory, from the foundational works of Arrow K. Arrow (1951) to contemporary developments, has consistently faced the challenge of designing voting systems that are both robust and representative. Voting constitutes a fundamental pillar of the functioning of democratic societies. It arises in numerous contexts of everyday life, ranging from political elections to collective decision-making in school settings. As emphasized in Barrot (2016), most of us have already participated in an electoral

process, sometimes as early as primary school, during the election of class representatives. Moreover, reference Savadogo et al. (2019) highlights the importance of analyzing different voting systems in order to better understand their mechanisms and implications.

In a context where post-electoral disputes are increasing, particularly in Africa, the question of the adequacy of traditional preference aggregation methods for large-scale elections arises acutely, both from a computational and a democratic standpoint. The emergence of the VDEPCI method Savadogo (2023) is part of this quest by proposing an innovative approach combining mathematical metrics and democratic theory. Based on the computation of distances with respect to an ideal candidate, this method offers a promising alternative to traditional systems while meeting modern criteria of electoral robustness.

The purpose of an election is to enable members of a society to choose their representatives or to make important decisions on specific issues. It provides a means for citizens to participate in the democratic process by expressing their will through their vote. The literature provides various examples in which voting is used to synthesize individual preferences into a collective decision, adapted to each situation. These scenarios vary according to the scope of the election, the number of voters, and the number of candidates involved. Aggregation methods differ between large-scale elections such as presidential, legislative, or municipal elections, and more limited elections such as those of class representatives or trade union bodies. Faced with this diversity of situations, preference aggregation methods must adapt to the specificities of each context. Large-scale elections, in particular, pose significant computational challenges that require efficient algorithmic approaches Brandt et al. (2016).

Recent work has emphasized the importance of developing efficient and scalable voting algorithms capable of handling elections involving a large number of voters and candidates Boehmer et al. (2024); Arora et al. (2024). This highlights the necessity of optimized implementations to ensure both computational efficiency and practical applicability in modern electoral contexts.

Our work specifically aims to propose an optimized algorithmic implementation of the VDEPCI method in order to meet the growing demand for large-scale elections, with the objective of simplifying computations and saving time. To this end, we will first present a review of the literature, followed by a presentation of the optimized algorithmic implementation, several applications and results, and finally a conclusion.

2 State of the Art

2.1 Some Social Choice Functions

A social choice function R.-B. O. Ngoie (2016) is a function $\mathcal{C}$ that associates with any situation (election) a non-empty subset of A, that is:

$$\begin{aligned} \mathcal{C} : \{\text{situations}\} &\longrightarrow \mathcal{K} \subset A \\ (A, p) &\longmapsto \mathcal{C}(A, p) \subseteq A \text{ with } \mathcal{C}(A, p) \neq \emptyset \end{aligned}$$

For a given situation, there may be several ways to aggregate individual preferences. However, it is preferable that the chosen aggregation method (social choice function) satisfies a certain number of properties reflecting the notion of democracy, namely: universality, non-dictatorship, unanimity, anonymity, monotonicity, consistency, ...
Examples of some social choice functions:

1. Single-round plurality voting
   Each voter votes for at most one candidate. The candidate who obtains the largest number of votes (relative majority) is the winner.

2. Two-round plurality voting
   Each voter votes for at most one candidate. If a candidate obtains an absolute majority ($> 50\%$), they are elected. Otherwise, a second round is organized between the two candidates who received the highest number of votes. This rule is the most widely used in the world for direct elections (by universal suffrage), but it has received very little attention from specialists in Social Choice Theory. Another name: two-round majority voting.
3. Approval voting
   Each voter approves as many candidates as they wish. The candidate who obtains the largest number of approvals is elected. Another name: approval voting (AV). See Baujard & Igersheim (2007); Laslier & Sanver (2010).
4. Score voting
   Each voter assigns to each candidate as many points as they wish between zero and, for example, 10 points. The elected candidate is the one who receives the largest total number of points. Other names: utilitarianism, score voting. See K. J. Arrow et al. (2010).

2.2 Majority Judgment

This section is a reminder of notions partly drawn from R.-B. M. Ngoie et al. (2014); Balinski & Laraki (2012). A common language is defined as a set of grades or evaluations $\mathcal{L} = \{g_1, g_2, \ldots, g_k\}$ strictly ordered by $\ll\succ\gg$ such that $g_1 \succ g_2 \succ \cdots \succ g_k \quad (g_i \succsim g_j$ if and only if $g_i \succ g_j$ or $g_i \sim g_j)$.
Let $f : \mathcal{L}^n \times A \longrightarrow \mathcal{L}$, where A is the set of candidates and n is the number of judges or voters. Let c_i be a candidate or competitor with grades $g_{i1}, g_{i2}, \ldots, g_{in}$, where $g_{i1} \succsim g_{i2} \succsim \cdots \succsim g_{in}$, assigned by the n voters. Then the majority evaluation or majority grade $f^{maj}(c_i)$ is defined by:

$$f^{maj}\left((g_{i1}, g_{i2}, \ldots, g_{in}), c_i\right) = \begin{cases} \pi_{\frac{n+1}{2}}\left(g_{i1}, g_{i2}, \ldots, g_{in}\right), & \text{if } n \text{ is odd,} \\ \pi_{\frac{n+2}{2}}\left(g_{i1}, g_{i2}, \ldots, g_{in}\right), & \text{if } n \text{ is even.} \end{cases}$$

Where π_j denotes the orthogonal projection onto the j^{th} component of the vector $(g_{i1}, g_{i2}, \ldots, g_{in})$. For example, if 5 judges assign the grades $4, 8, 7, 9, 5$ to c_i,

$$f^{maj}\left((9, 8, 7, 5, 4), c_i\right) = \pi_3(9, 8, 7, 5, 4) = 7.$$

And if 8 judges assign the grades $9, 7, 3, 6, 5, 4, 5, 8$ to c_i,

$$f^{maj}\left((9, 8, 7, 6, 5, 5, 4, 3), c_i\right) = \pi_5(9, 8, 7, 6, 5, 5, 4, 3) = 5.$$

When the majority grades of two candidates are different, the candidate who obtains the higher majority grade is ranked ahead of the other. The majority ranking $\succ_{\text{maj}}$ between two candidates evaluated by the same jury is determined by a repeated application of the majority grade:

- If $f^{maj}\left((g_{i1}, g_{i2}, \ldots, g_{in}), c_i\right) \succ f^{maj}\left((g_{j1}, g_{j2}, \ldots, g_{jn}), c_j\right)$, then $c_i \succ_{\text{maj}} c_j$.
- If $f^{maj}\left((g_{i1}, g_{i2}, \ldots, g_{in}), c_i\right) \sim f^{maj}\left((g_{j1}, g_{j2}, \ldots, g_{jn}), c_j\right)$, that is, if the majority evaluations of c_i and c_j yield identical grades, then this grade is removed from the list of grades of each candidate and the procedure is repeated.

2.3 Description of the VMAVA Method

The main developments presented in this section are drawn from Savadogo & Some (2020). In what follows, we present the voting method based on approval voting and the arithmetic mean (Voting Method based on Approval Voting Arithmetic Mean), also called VMAVA. Consider a set $\mathcal{M}$ of m candidates for a given election with $m \geq 2$, and a set $\mathcal{A}$ of n voters with $n \geq 2$.

Each voter assigns to a given partition of the set $\mathcal{M}$ the following ordered preferences: 1^{st} choice, 2^{nd} choice, 3^{rd} choice, 4^{th} choice. Thus, they assign to each subset of the partition the scores $4, 3, 2, 1$, respectively. The method first consists in computing the arithmetic mean of the scores given by each voter j to a given candidate, and then using three sets $Gsup_j$, $Gmoy_j$, and Ginf_j defined as follows:

1. $Gsup_j$ consists of the candidates whose score assigned by the voter is strictly greater than the arithmetic mean.
2. $Gmoy_j$ consists of the candidates whose score assigned by the voter is equal to the arithmetic mean.
3. $Ginf_j$ consists of the candidates whose score assigned by the voter is strictly less than the arithmetic mean.

Subsequently, for each voter V_j, we retain the candidates that belong to the set $Gsup_j$. We then compute the intersection of all the sets $Gsup_j$:

- If this intersection is reduced to a singleton, that is, a single candidate, then this candidate is the winner.
- If this intersection contains more than one candidate, we consider the arithmetic mean of their scores and select as the winner the candidate with the highest arithmetic mean. More generally, if we have n voters, with n even (resp. odd), the winner is the candidate belonging to at least $\frac{n}{2}+1$ (resp. $E\left(\frac{n}{2}\right)+1$, where $E(.)$ denotes the integer part function) of the sets $Gsup_j$.
- If this intersection is empty, then the same procedure is repeated by considering the intersection of the sets $Gmoy_j$. However, note that these candidates will be moderately appreciated by the voters. If the intersection of the sets $Gmoy_j$ is empty, then it would be desirable to reconsider the electoral process, at the risk of using the sets Ginf_j, which would lead to the election of a candidate not at all appreciated by the majority or by all of the voters.

2.4 VDEPCI Method

2.4.1 Description of the VDEPCI Method

The VDEPCI method is built, on the one hand, on the use of approval voting and, on the other hand, on the use of an ideal candidate.
The method is structured according to the following steps:
Step 1: We consider:

- A finite set of candidates: $A = \{c_1, c_2, \ldots, c_m\}$ with $m \geq 2$
- A finite set of voters: $N = \{V_1, V_2, \ldots, V_n\}$ with $n \geq 2$
- Each voter V_j assigns an ordered partition of A into four levels of preference:
  1^{st} choice, 2^{nd} choice, 3^{rd} choice, 4^{th} choice. Thus, they assign to each subset of the partition the scores $4, 3, 2, 1$, respectively.

Step 2: We determine an ideal candidate, which is a hypothetical candidate who would receive the maximum score of 4 from all voters.
Step 3: For each candidate $c_i \in A$, we compute:

- Manhattan distance:
$$d^1_{c_i} = \sum_{j=1}^{n} |x_{ij} - 4|$$
- Euclidean distance (used in case of a tie):
$$d^2_{c_i} = \sqrt{\sum_{j=1}^{n} (x_{ij} - 4)^2}$$

Step 4: We select the best candidate or winner:

- We search for the candidate(s) minimizing $d^1_{c_i}$:

$$\mathcal{W}_1 = \underset{c_i \in A}{\operatorname{argmin}} \left(d^1_{c_i}\right)$$

If $|\mathcal{W}_1| = 1$, the unique candidate is declared the winner.

- In case of a tie, i.e., $|\mathcal{W}_1| > 1$
Among the candidates in $\mathcal{W}_1$, we retain the one minimizing $d^2_{c_i}$:

$$\text{Winner} = \underset{c_k \in \mathcal{W}_1}{\operatorname{argmin}} \left(d^2_{c_k}\right)$$

2.4.2 Presentation of the Applications of the VDEPCI Method

Example 1: 4 Candidates and 5 Voters
The examples presented in this section are taken from Savadogo (2023).

Table 1: Matrix of Choices

Notes	4	3	2	1
Choices	1st choice	2nd choice	3rd choice	4th choice
V_1	$\{c_2, c_4\}$	$\{c_1\}$	$\{c_3\}$	
V_2	$\{c_1\}$	$\{c_3\}$	$\{c_4\}$	$\{c_2\}$
V_3	$\{c_3\}$	$\{c_1\}$	$\{c_2, c_4\}$	
V_4	$\{c_1\}$	$\{c_2, c_4\}$	$\{c_3\}$	
V_5	$\{c_1\}$	$\{c_3\}$		$\{c_2, c_4\}$

Which corresponds to the following voting matrix:

Table 2: Note Matrix

	c_1	c_2	c_3	c_4
V_1	3	4	2	4
V_2	4	1	3	2
V_3	3	2	4	2
V_4	3	2	1	2
V_5	4	1	3	1

From Table (2), we compute $d^1_{c_i}$, the distance of candidate c_i from the ideal candidate, as follows:

$$d^1_{c_1} = |3-4| + |4-4| + |3-4| + |3-4| + |4-4| \quad \Longrightarrow d^1_{c_1} = 3$$
$$d^1_{c_2} = |4-4| + |1-4| + |2-4| + |2-4| + |1-4| \quad \Longrightarrow d^1_{c_2} = 10$$
$$d^1_{c_3} = |2-4| + |3-4| + |4-4| + |1-4| + |3-4| \quad \Longrightarrow d^1_{c_3} = 7$$
$$d^1_{c_4} = |4-4| + |2-4| + |2-4| + |2-4| + |1-4| \quad \Longrightarrow d^1_{c_4} = 9$$

Thus, c_1 is the best candidate or the winner.

Example 2: 5 Candidates and 7 Voters
The examples presented in this section are taken from Koumbèbarè Kambiré & Nikiéma (2023).

Table 3: Matrix of Choices

Notes	4	3	2	1
Choices	1st choice	2nd choice	3rd choice	4th choice
V_1	$\{c_2, c_5\}$	$\{c_4\}$	$\{c_1, c_3\}$	
V_2	$\{c_5\}$	$\{c_4\}$		$\{c_1, c_2, c_3\}$
V_3		$\{c_4\}$	$\{c_1, c_3\}$	$\{c_2, c_5\}$
V_4	$\{c_2\}$	$\{c_4, c_5\}$		$\{c_1, c_3\}$
V_5	$\{c_3\}$	$\{c_1, c_2, c_4, c_5\}$		
V_6	$\{c_5\}$	$\{c_1, c_2, c_3, c_4\}$		
V_7	$\{c_1\}$	$\{c_3, c_4\}$		$\{c_2, c_5\}$

We thus obtain the following voting matrix:

Table 4: Note Matrix

	c_1	c_2	c_3	c_4	c_5
V_1	2	4	2	3	4
V_2	1	1	1	3	4
V_3	2	1	2	3	1
V_4	1	4	1	3	3
V_5	3	3	4	3	3
V_6	3	3	3	3	4
V_7	4	1	3	3	1

From Table (4), we compute $d^1_{c_i}$, the distance of candidate c_i from the ideal candidate, as follows:

$$d^1_{c_1} = 2+3+2+3+1+1+0 \quad \Longrightarrow d^1_{c_1} = 12$$
$$d^1_{c_2} = 0+3+3+0+1+1+3 \quad \Longrightarrow d^1_{c_2} = 11$$
$$d^1_{c_3} = 2+3+2+3+0+1+1 \quad \Longrightarrow d^1_{c_3} = 12$$
$$d^1_{c_4} = 1+1+1+1+1+1+1 \quad \Longrightarrow d^1_{c_4} = 7$$
$$d^1_{c_5} = 0+0+3+1+1+0+3 \quad \Longrightarrow d^1_{c_5} = 8$$

Thus, c_4 is the best candidate or the winner.

3 Implementation of the VDEPCI Method

Although the approach proposed in this work is inspired by that of Koumbèbarè Kambiré & Nikiéma (2023), who implemented the VMAVA (Voting Method based on Approval Voting Arithmetic Mean) method in MATLAB, our approach differs through a Python implementation, providing greater flexibility

and computational efficiency suitable for processing elections involving a large number of voters and candidates. It follows the same perspective of analyzing decision-support methods. The Python implementation that we propose constitutes an independent alternative, based on a distinct architecture and specific algorithmic choices. This approach aims not only to provide a more accessible and flexible solution thanks to the advantages of the Python language, but also to explore new possibilities for optimization and integration with recent libraries. It thus makes it possible to better meet current needs in the processing of electoral data and large-scale simulations.

3.1 Algorithmic Procedure of the VDEPCI Method

The VDEPCI method relies on a rigorous procedure composed of well-defined steps, making it possible to systematically identify the winner based on the preferences expressed by the voters.

- Data loading and validation
  This first step consists in importing a structured `.csv` file containing the evaluations of the candidates by all voters. From this file, a `NumPy` matrix representing the votes as well as the list of candidates are extracted. A verification of the consistency and integrity of the data is also performed.

- Computation of the metrics
  For each candidate, two types of distances are computed: the Manhattan distance ($d^1_{c_i}$) and a second distance ($d^2_{c_i}$), used in the case of a tie. These distances make it possible to measure the overall deviation between individual preferences and a hypothetical consensus position.

- Selection of the winner
  The candidate with the smallest value of $d^1_{c_i}$ is initially selected. If several candidates exhibit the same minimal distance, a second selection is then carried out using the values of $d^2_{c_i}$ in order to break the tie.

3.2 Implementation in Python

Python is a general-purpose programming language operating within an object-oriented approach Keita (2017). The implementation of the VDEPCI method was carried out in the Python language due to its richness in scientific libraries and its efficiency in processing matrix-based data. The language allows for easy manipulation of data structures, fast computation of metrics, as well as possible visualization of results.

3.2.1 Pseudo-code of the Method

This pseudo-code presents the essential steps of the VDEPCI method, from data import to the final selection of the winner.

The code and data are available on GitHub: https://github.com/bamogohamado/VDEPCI.git.

Algorithm 1: Pseudo-code of the VDEPCI Method

Input: CSV file containing the scores assigned to candidates by voters
Output: Winning candidate

Load the vote matrix X from the file
Extract the list of candidates
for *each candidate* c_i **do**
    Compute $d^1_{c_i} = \sum_{j=1}^{n} |x_{ji} - 4|$
Identify the set C_1 of candidates with the minimum value of d^1
if $|C_1| = 1$ **then**
    Return the candidate in C_1 as the winner
else
    for *each candidate* $c_i \in C_1$ **do**
        Compute $d^2_{c_i} = \sum_{j=1}^{n} (x_{ji} - 4)^2$
    Identify the set C_2 of candidates with the minimum value of d^2
    if $|C_2| = 1$ **then**
        Return the candidate in C_2 as the winner
    else
        Randomly select a candidate from C_2
        Return this candidate as the winner

3.2.2 Python Code: Complete Implementation of the Method

We will first create a function **LoadData** that loads the data, then two functions, **DistanceManhattan** and **DistanceEuclidean**, which compute the Manhattan and Euclidean distances, respectively, and finally a function **VdpeciMethod** that returns the best candidate or the winner.

Function 1 : Data

```python
def LoadData(csv_file):
    # Read the CSV file containing voter preferences
    df = pd.read_csv(csv_file)

    # Extract candidate names from the columns
    candidates = df.columns.tolist()

    # Convert the data into a vote matrix (voters × candidates)
    vote_matrix = df.values

    # Return the vote matrix and the list of candidates
    return vote_matrix, candidates
```

Function 2 : Distance from Manhattan

```
def DistanceManhattan(vote_matrix):
    # Compute the Manhattan distance between each candidate and the ideal candidate
    # The ideal value is assumed to be 4 for each voter
    d1 = np.sum(np.abs(vote_matrix - 4), axis=0)

    # Return the distance vector (one value per candidate)
    return d1
```

Function 3 : Euclidean distance

```
def DistanceEuclidean(vote_matrix):
    # Compute the squared Euclidean distance between each candidate and the ideal candidate.
    # The ideal value is assumed to be 4 for each voter
    d2 = np.sum((vote_matrix - 4) ** 2, axis=0)

    # Return the distance vector (one value per candidate)
    return d2
```

Function 4 : VDEPCI Method

```
def Vdepci_method(csv_file):

    # Step 1: Load data
    X, candidates = LoadData(csv_file)

    # Step 2: Distance from Manhattan
    d1 = DistanceManhattan(X)
    min_d1 = np.min(d1)
    min_d1_indices = np.where(d1 == min_d1)[0]

    # Select winner if no tie
    if len(min_d1_indices) == 1:
        winner = candidates[min_d1_indices[0]]
    else:
        # Step 3: Tie - Euclidean distance
        d2 = DistanceEuclidean(X)
        d2_tied = d2[min_d1_indices]
        min_d2 = np.min(d2_tied)
        min_d2_indices = np.where(d2_tied == min_d2)[0]

        # Select final winner
        if len(min_d2_indices) == 1:
            winner = candidates[min_d1_indices[min_d2_indices[0]]]
```

```
    # Output: Display distances and winner
    print("d1_distances:", dict(zip(candidates, d1)))
    if len(min_d1_indices) > 1:
        dd = {candidates[min_d1_indices[i]]: d2_tied[i] for i in range(len(d2_tied))}
        print("d2 distances (ties):", dd)
    print(" The winning candidate is:", winner)
```

Program : Main program

```
# Main program
Vdepci_method(csv_file)
```

3.3 Algorithmic Complexity

3.3.1 Theoretical Study

The objective of this theoretical study is to evaluate the algorithmic cost (in time) of each phase of the method as a function of the number of voters (n) and the number of candidates (m).

- **Data loading:** This operation requires scanning all the elements of the array, i.e., $n \times m$ values, which yields a complexity of $\mathcal{O}(n \times m)$.
- **Computation of distance d_1:** For each candidate (m columns), all votes (n rows) are scanned while computing the sum of absolute deviations, resulting in a complexity of $\mathcal{O}(n \times m)$.
- **Identification of the candidate(s) with the smallest distance d_1**
  The minimum is searched in an array of size m, leading to a complexity of $\mathcal{O}(m)$.
- **Computation of distance d_2:** This step has the same complexity as that of distance d_1, namely $\mathcal{O}(n \times m)$.
- **Final selection:** This operation consists in selecting one element from a list of size 1, and therefore has a complexity of $\mathcal{O}(1)$.

The most computationally expensive operations are those that require a full traversal of the voting matrix, in particular the computation of d_1 and d_2. These two operations exhibit an algorithmic complexity of $\mathcal{O}(n \times m)$ in the worst case. Consequently, the overall complexity of the VDEPCI method is linear with respect to the size of the voting matrix, i.e., $\mathcal{O}(n \times m)$.

3.3.2 Experimental Study

In this section, an experimental study is conducted to complement the theoretical analysis with concrete performance measurements. The objective is to measure the actual execution times of the VDEPCI method as a function of the number of voters (n) and the number of candidates (m). The experiments are performed on a set of synthetic datasets whose size is systematically varied, with the number of voters ranging from 100 to 10000 and the number of candidates ranging from 3 to 20. These intervals were selected to cover scenarios ranging from small to medium-scale elections, while allowing an evaluation of the method's behavior as the problem size progressively increases. This experimental configuration enables a systematic analysis of the influence of the parameters n and m on execution time and allows assessing the scalability of the method. The obtained results empirically confirm that the computational complexity is indeed linear, that is, $\mathcal{O}(n \times m)$.
The following methodology is adopted:

- **Generation:** Random voting matrices are simulated for different sizes.
- **Execution time measurement:** The time required for each call to the **VdpeciMethod** function is measured.
- **Plotting the curves:** Execution-time curves are plotted as a function of n for fixed m, and as a function of m for fixed n.

Example: Let us generate datasets with the following sizes:

$$\text{Number of voters} = [100, 500, 1000, 2000, 4000, 10000]$$
$$\text{Number of candidates} = [3, 5, 7, 10, 15, 20]$$

The complete experimental study of the VDEPCI method yields the graph below:

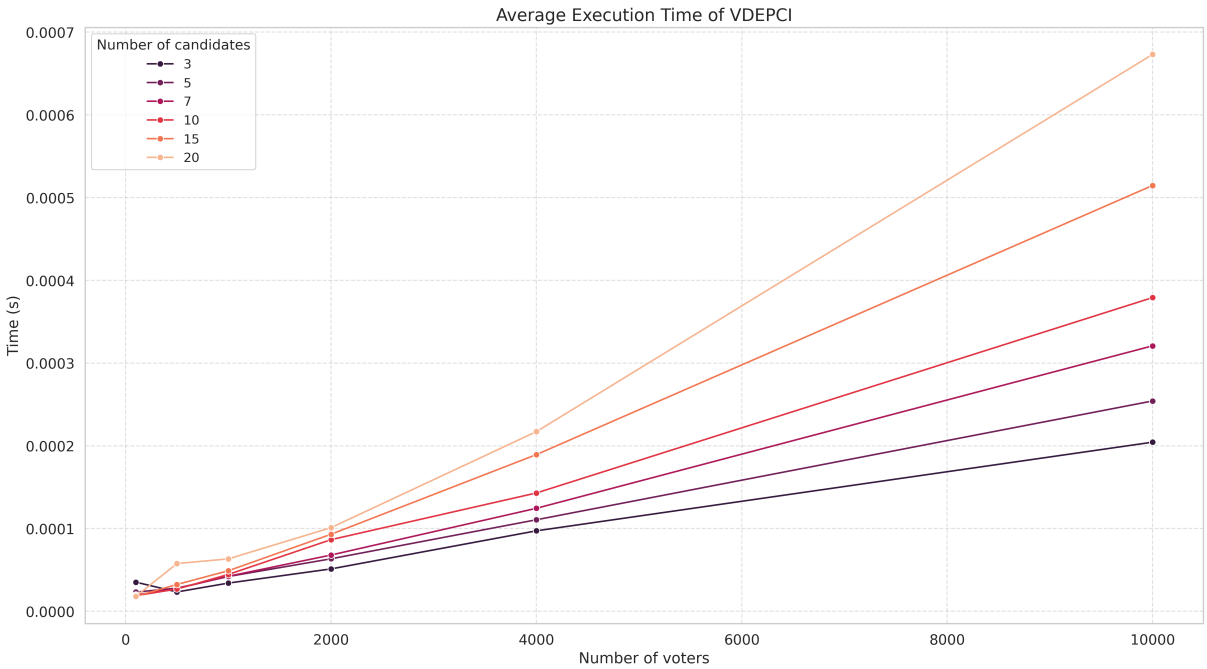

Figure 1: Experimental complexity graph

It can be observed that the execution time increases proportionally to $n \times m$.
This confirms the theoretical complexity presented above, thus showing that the method is suitable for elections involving several thousand voters.

4 Applications

4.1 Example 1

Let us revisit Example 1 [2] presented on page [5].
The analysis of the input data by our implementation of the VDEPCI method produces the following results, structured and displayed in the output console:

- **Data loading and call to the VdepciMethod function**

```
# Main program
csv_file = '/Users/macbookpro/Desktop/VDEPCI/Example1.csv'
vdepci_method(csv_file)
```

- **Output**

```
In [1]: %runfile /Users/macbookpro/Desktop/VDEPCI/Vdepci_method.py --wdir
d1_distances: {'C1': 3, 'C2': 10, 'C3': 7, 'C4': 9}
 The winning candidate is: C1
```

After program execution, candidate c_1 emerges as the winner. This is the same winner as the one obtained in Section [2].

4.2 Example 2

Let us revisit Example 2 [4] presented on page [6].
By entering these data into our program, we obtain the following result in the console:

- **Loading**

```
# Main program
csv_file = '/Users/macbookpro/Desktop/VDEPCI/Example2.csv'
vdepci_method(csv_file)
```

- **Output**

```
In [2]: %runfile /Users/macbookpro/Desktop/VDEPCI/Vdepci_method.py --wdir
d1_distances: {'C1': 12, 'C2': 11, 'C3': 12, 'C4': 7, 'C5': 8}
 The winning candidate is: C4
```

After program execution, candidate c_4 is identified as the winner. This is the same winner as the one obtained in Section [4].

4.3 Example 3

In this example, we simulate a random voting matrix with 500 voters and 9 candidates.
These data are represented in Fig [2] below:

Figure 2: Vote distribution matrix

- **Loading and processing**

```
# Main program
csv_file = '/Users/macbookpro/Desktop/VDEPCI/Example3.csv'
vdepci_method(csv_file)
```

- **Output**

```
In [3]: %runfile /Users/macbookpro/Desktop/VDEPCI/Vdepci_method.py --wdir
d1_distances: {'C1': 714, 'C2': 744, 'C3': 770, 'C4': 650, 'C5': 775, 'C6': 731, 'C7': 734, 'C8': 776, 'C9': 764}
 The winning candidate is: C4
```

The execution of the program reveals that candidate c_4 is the winner.

4.4 Example 4

In this case study, we simulate a random preference matrix in order to model the electoral behavior of 1000 voters facing a set of 15 candidates.
The graphical representation of these data is given in Fig [3] below:

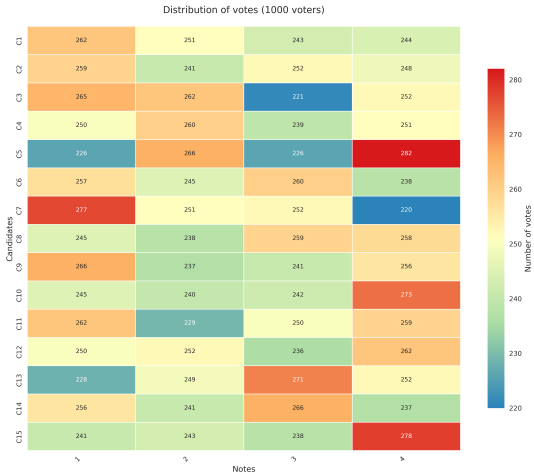

Figure 3: Vote distribution matrix

- **Loading and processing**

```
# Main program
csv_file = '/Users/macbookpro/Desktop/VDEPCI/Example4.csv'
tps1 = time.time()
vdepci_method(csv_file)
tps2 = time.time()
print(f"Execution time is {tps2 - tps1} seconds")
```

- **Output**

```
In [4]: %runfile '/Users/macbookpro/Documents/Mémoire/VDEPCI/vdepci_method.py' --wdir
d1_distances: {'C1': 1531, 'C2': 1511, 'C3': 1540, 'C4': 1509, 'C5': 1436, 'C6': 1521, 'C7': 1585, 'C8': 1470, 'C9': 1513,
'C10': 1457, 'C11': 1494, 'C12': 1490, 'C13': 1453, 'C14': 1516, 'C15': 1447}
The winning candidate is: C5
Execution time is 0.025887012481689453 seconds
```

After program execution, the results indicate that candidate c_5 obtains the majority, thereby being designated as the winner of the election. The execution time of the program, evaluated at only 0.026 seconds, demonstrates its efficiency. This fast execution, even for a dataset including 1000 voters and 15 candidates, indicates that the implemented algorithm is well optimized and suitable for large-scale use.

5 Discussion

The experimental results confirm the high computational efficiency of the VDEPCI implementation. The observed execution time of 0.026 seconds for 1000 voters and 15 candidates shows that the computational cost per voter-candidate pair is very low. This result is consistent with the theoretical linear time complexity, $\mathcal{O}(n \times m)$, which implies a proportional increase in execution time with the number of voters and candidates. This property ensures excellent scalability. In particular, even for several thousands, or tens of thousands of voters, the expected execution time remains on the order of a few fractions of a second. This demonstrates that the implementation is capable of efficiently

processing elections involving a large number of participants, thereby confirming its applicability to large-scale datasets and real-world voting contexts.

6 Conclusion

The implementation of the VDEPCI method in Python represents a concrete step forward in the operational deployment of voting methods adapted to collective decision-making contexts. This approach made it possible to translate a theoretical formalism into a functional tool capable of efficiently processing large-scale simulated electoral data. The choice of the Python language, due to its simplicity, readability, and the abundance of its libraries, facilitated development while offering opportunities for extension toward more complex applications.
This work is part of a broader research trend on the operationalization of voting methods, as illustrated by the study *Implementation of a Voting Method Based on Mean-Deviation Evaluation for a Large-Scale Election* Yiogo & Savadogo (2025), which also proposes a concrete implementation in an extended electoral context. These contributions confirm the relevance of computational approaches for analyzing and simulating collective behavior in voting situations.
Our work opens the way to future investigations. First, the optimization of the algorithm can be further explored, notably by integrating specialized libraries to accelerate processing on large volumes of data. Next, it would be relevant to extend the implementation to other voting rules (such as Borda, Condorcet, etc.) in order to compare their performance and outcomes in different contexts. Finally, the development of a graphical user interface or a web application would make the tool accessible to a non-technical audience, thereby facilitating its use in pedagogical or real decision-making contexts.

Acknowledgment

A brief acknowledgement section may be given after the conclusion section just before the references. The acknowledgments of people who provided assistance in manuscript preparation, funding for research, etc. should be listed in this section.

References

Arora, A., Eppstein, D., & Huynh, R. L. (2024). Fast schulze voting using quickselect. *arXiv preprint arXiv:2411.18790*.

Arrow, K. (1951). 1963. Social choice and individual values. *Revised edition. New Haven*.

Arrow, K. J., Sen, A., & Suzumura, K. (2010). *Handbook of social choice and welfare* (Vol. 2). Elsevier.

Balinski, M. L., & Laraki, R. (2012). Jugement majoritaire vs. vote majoritaire.

Barrot, N. (2016). *Sur les aspects computationnels du vote par approbation* (Unpublished doctoral dissertation). Université Paris sciences et lettres.

Baujard, A., & Igersheim, H. (2007). *Expérimentation du vote par note et du vote par approbation lors de l'élection présidentielle française du 22 avril 2007* (Unpublished doctoral dissertation). Centre d'analyse stratégique.

Boehmer, N., Faliszewski, P., Janeczko, Ł., Kaczmarczyk, A., Lisowski, G., Pierczyński, G., ... Was, T. (2024). Guide to numerical experiments on elections in computational social choice. *arXiv preprint arXiv:2402.11765*.

Brandt, F., Conitzer, V., Endriss, U., Lang, J., & Procaccia, A. D. (2016). *Handbook of computational social choice*. Cambridge University Press.

Keita, M. (2017). *Data science sous python: Algorithme, statistique, dataviz, datamining et machine-learning [data science with python: Algorithm, statistics, dataviz, datamining and machine-learning]* (Tech. Rep.). University Library of Munich, Germany.

Koumbèbarè Kambiré, Z. S., & Nikiéma, F. (2023). Implementation of the vmava method in order to make applications with a large number of candidates and voters. *Pure and Applied Mathematics Journal*, *12*(3), 49-58.

Laslier, J.-F., & Sanver, M. R. (2010). The basic approval voting game. *Handbook on approval voting*, 153–163.

Ngoie, R.-B. M., Savadogo, Z., & Ulungu, B. E.-L. (2014). Median and average as tools for measuring, electing and ranking: new prospects.

Ngoie, R.-B. O. (2016). *Nouvelle approche basée sur la moyenne et la médiane en théorie du choix social et analyse multicritère* (phdthesis, Université Pédagogique Nationale (UPN)). Retrieved 2025-03-26, from https://theses.hal.science/tel-01521587

Savadogo, Z. (2023). VOTING METHOD BASED ON A DISTANCE ASSESSMENT OF PREFERENCES IN RELATION TO THE IDEAL CANDIDATE. *Universal Journal of Mathematics and Mathematical Sciences*, *18*(1), 99–119. Retrieved 2025-05-08, from https://www.pphmjopenaccess.com/index.php/ujmms/article/view/153

Savadogo, Z., Compaoré, A., & Ouedraogo, P. O. F. (2019). Voting method based on an average gap assessment. *European Journal of Pure and Applied Mathematics*, *12*(3), 1176–1186.

Savadogo, Z., & Some, B. (2020). Voting method based on approval voting and arithmetic mean. *International Journal of Applied Mathematical Research*, *9*(1), 1–8.

Yiogo, H., & Savadogo, Z. (2025). Implementation of a voting method based on mean-deviation evaluation for a large-scale election. *Pure and Applied Mathematics Journal*, *14*(2), 13-23. Retrieved from https://doi.org/10.11648/j.pamj.20251402.11 doi: 10.11648/j.pamj.20251402.11