Journal of Big Data

RESEARCH Open Access

New distributed-topsis approach for multi-criteria decision-making problems in a big data context

Loubna Lamrini[1*], Mohammed Chaouki Abounaima[1] and Mohammed Talibi Alaoui[1]

*Correspondence:
loubna.lamrini@usmba.ac.ma

[1] Laboratory of Intelligent Systems and Applications, Faculty of Sciences and Technologies, Sidi Mohammed Ben Abdellah University, Fez, Morocco

Abstract

Nowadays, the online environment is extra information-rich and allows companies to offer and receive more and more options and opportunities in multiple areas. Thus, decision-makers have abundantly available alternatives to choose from the best one or rank from the most to the least preferred. However, in the multicriteria decision-making field, most tools support a limited number of alternatives with as narrow criteria as possible. Decision-makers are forced to apply a screening or filtering method to reduce the size of the problem, which will slow down the process and eliminate some potential alternatives from the rest of the decision-making process. Implementing MCDM methods in high-performance parallel and distributed computing environments becomes crucial to ensure the scalability of multicriteria decision-making solutions in Big Data contexts, where one can consider a vast number of alternatives, each being described on the basis of a number of criteria.

In this context, we consider TOPSIS one of the most widely used MCDM methods. We present a parallel implementation of TOPSIS based on the MapReduce paradigm. This solution will reduce the response time of the decision-making process and facilitate the analysis of the robustness and sensitivity of the method in a high-dimension problem at a reasonable response time.

Three multicriteria analysis problems were evaluated to show the proposed approach's computational efficiency and performance. All experiments are carried out within GCP's Dataproc, a service allowing the execution of Apache Hadoop and Spark tasks in Google Cloud. The results of the tests obtained are very significant and promising.

Keywords: Multi-criteria decision making (MCDM), TOPSIS, MapReduce, Apache Spark, Apache Hadoop, PySpark, Dataproc, Google cloud platform (GCP)

Introduction

With the digital age and the phenomenon of BIG DATA, decision-making problems occur in everyone's daily life. We are faced with a large number of options, for example: buy which product from which supplier, target which customer, hire which candidate, etc. MCDM methods help decision-makers structure a complex decision-making

problem with various dimensions and explore promising courses of action to converge towards optimal solutions that balance the different concerns of decision-makers.

Multi-criteria decision support is an alternative to traditional optimization methods based on the definition of a single function that reflects the consideration of several criteria [1]. The advantage of multi-criteria methods is to consider a set of criteria reflecting different dimensions of the decision problem posed in the decision-making process. It is not a question of looking for an optimum but a compromise solution with various forms: choice, ranking or classification. Several methods exist in the literature. As part of this work, given its fame and importance in this field, we are interested in the TOPSIS method (Technique for Order Preference by Similarity to the Ideal Solution) [2].

Different software packages have been developed for several years to solve MCDM problems with TOPSIS. However, most available software has limitations in terms of the alternatives and the criteria to consider [3]. For example, with an exclusive commercial license, the Triptych package for TOPSIS analysis [4] requires Microsoft Excel and limits the number of alternatives to 200. SANNA and TOPSIS Solver software [3, 4] also need Microsoft Excel to run and limit the number of alternatives to 200. In the case of large datasets, decision makers (DMs) have to exert more effort to reduce the size of the set of alternatives and adapt it to the capacity imposed by the MCDM software programs. However, little research has been conducted on how big data tools can be applied to overcome this limitation.

Data is constantly increasing with the popularization of means of communication, Internet of Things (IoT), social networks, etc. Thus, decision-making has become a complex problem due to the abundance of alternatives and the wide variety of criteria. To circumvent this problem, researchers propose new models coupling MCDM methods and machine learning and artificial intelligence approaches [5–9]. Nevertheless, these models need robust frameworks, techniques, and algorithms for big data problems. MCDM methods and big data fill a very valuable research gap. Although there are several textbooks and research papers in the field of MCDM [10], there is no framework for MCDM methods in the context of emerging big data.

Several solutions have been proposed to cope with the increase in data flow and the large number of alternatives to be processed. In [11–13], screening techniques were proposed to eliminate, from the beginning of the decision-making process, the alternatives deemed irrelevant. Screening makes it possible to reduce the size of the decision matrix, but in the case of a ranking process, the filtered alternatives will not appear in the final list, which contains the sorted alternatives. Moreover, the filtering operation risks eliminating relevant alternatives from the beginning of the decision-making process.

To reduce the number of alternatives to be processed by MCDM methods, service providers are compelled to devote more effort to simplifying their offerings and facilitating the operation of choice. For example, the famous American company Expedia, which operates several online travel agencies, allows hotels to present a limited number of room types instead of showing all of their available room types [14]. However, with larger choice sets, customers have a better chance of matching their purchases to their preferences and maintaining their flexibility [15].

In this paper, we propose a parallel implementation to address this problem. A proven effective parallel computing paradigm is MapReduce [16]. It is a massively parallel

programming model suitable for processing very large amounts of data. Programs adopting this model are automatically parallelized and executed on clusters. Indeed, distributed parallelism has always been a possibility to meet this performance requirement. It is even more true today with the massive presence of parallel computing and distributed storage clusters as well as the fall in cluster prices.

Implementing distributed parallel algorithms, such as our proposed approach, requires an infrastructure of distributed and connected machines in which the calculations are carried out in parallel. Today, the least expensive in terms of investment and the best-distributed computing solution is offered by cloud computing platforms. Indeed, cloud computing platforms have provided businesses with on-demand computing resources over the Internet, with pricing based on using those resources. Cloud computing remains a better solution for companies to avoid investing in buying, owning, and managing physical servers and data centers. Businesses and decision-makers can access technology services such as computing power, storage, and databases on demand with this mobile services solution. To test our proposed parallel distributed approach, we have chosen Google Cloud Platform (GCP), considered one of the most popular platforms, with the possibility of a free platform trial.

Three examples of multi-criteria analysis problems have been evaluated in this paper to prove the computational performance of the proposed Distributed-TOPSIS approach. As we will show in the experimentation section, the results of these evaluations are better and more significant for the case where the sets of alternatives become very large. Indeed, for a set of one million alternatives, we have obtained an execution time of 412 s for the sequential algorithm of the TOPSIS method, while for the proposed Distributed-TOPSIS approach, for the same set, we have obtained, with Dataproc clusters with two worker nodes and three worker nodes, an execution time of 121 and 84 s respectively.

The paper is composed of 5 sections. After the introduction in "Introduction" Section, the second one presents the background and the motivation for this study; it gives an overview of MCDM problems and their constraints. An analysis of the TOPSIS method with its computing complexity will be provided. We also present in this section the paradigm MapReduce and the importance of the implementation in Spark. "Proposed approach: Distributed TOPSIS approach and MapReduce calculation" Section outlines the algorithm proposed and its implementation according to the MapReduce paradigm. "Experimentation and discussion of results" Section demonstrates the algorithm's performance with real and generated data sets. Finally, in "Conclusions and Further Research" Section, we conclude this work by highlighting all the results obtained and giving some perspectives and future research scopes.

Background and motivation

In this section, we first briefly review the basic idea of MCDM methods. And then, we introduce TOPSIS, which is very powerful and widely used. Indeed, the main objective of this article is to find a solution for extending MCDM methods to apply them in a Big Data context where a large set of alternatives can be considered. The TOPSIS method is chosen to illustrate our new approach based on the Map-Reduce paradigm for distributed and parallel programming through a computing cloud. In addition, we opted for this method given its simplicity and the fact that it is considered one of the most widely

used MCDM methods. However, as specified in the perspective, the proposed approach will be generalized later for other MCDM methods.

The theoretical framework and methodological aspects of multi-criteria methods

Today's society gives us easy access to more information, products, and opportunities. When a decision-maker is given many sets of data, the quality of his decision is decreased because of the individual's limitation of resources to process all the information and optimally make the best decision [10]. The human brain cannot process data, which can lead to choice overload. A wide range of choices slows down and reduces decision-making processes (For example, when a client wants to book a hotel, buy a product online, or recruit a candidate).

The advent of modern information technology has been a primary driver of information overload on multiple fronts: quantity produced, ease of dissemination, and breadth of the audience reached. Technological factors have been further intensified by the rise of social media and the attention economy, which facilitates attention theft. In the age of connective digital technologies, the Internet culture, information overload is associated with over-exposure to information and input abundance of choices or alternatives. In most cases, too many options lead to suboptimal choice, a phenomenon known as choice overload. When people are faced with a large set of alternatives, they report less satisfaction and regret their decision [17].

MCDM methods assist decision-makers in decision-making by considering several parameters whose importance is different. For example, to buy a car, the customer often tends to find a compromise between quality and price. The selection criteria differ from person to person.

Decision matrix

Any MCDA problem considers a set A of n alternatives, consisting of candidate solutions, and a family F of m of criteria, the views points according to which the alternatives will be examined and compared. A set $\mathbb{W}$ of weights is also provided to consider the differences between the relative importance of criteria in decision-making. To summarize all the data of a decision problem, a matrix M, see Table 1, called a performance matrix, or decision matrix, is constructed [18].

– $F = \{g_1, \ldots, g_j, \ldots, g_m\}$ is the family of m criteria $m \geq 2$.

Table 1 Decision matrix

	Criteria					
$M=$	Weights	g_1	…	g_j	…	g_m
		w_1	…	w_j	…	w_m
		Min/Max	…	Min/Max	…	Min/Max
	a_1	$g_1(a_1)$	…	$g_j(a_1)$	…	$g_m(a_1)$
	…	…	…	…	…	…
	a_i	$g_1(a_i)$	…	$g_j(a_i)$	…	$g_m(a_i)$
	…	…	…	…	…	…
	a_n	$g_1(a_n)$	…	$g_j(a_n)$	…	$g_m(a_n)$

- $A = \{a_1, \ldots, a_i, \ldots, a_n\}$ is the set of alternatives.
- $W = \{w_1, \ldots, w_j, \ldots, w_m\}$ is the weight vector reflecting the relative importance of the criteria.
- $g_j(a_i)$ is the evaluation of the criterion g_j for the alternative a_i.
- Min indicates that the criterion is to be minimized.
- Max indicates that the criterion is to be maximized.

Real problems can be formulated using multi-criteria analysis methods according to three basic categories [1]:

- The choice problem (Pα) consists of MCDM problems in which the DMs must select a subset of alternatives evaluated as the best from a set A.
- The sorting or assignment problem corresponds to assigning each alternative to predefined ordered categories (Pβ). In such a problem, a set of categories must be a priori-defined, and actions are assigned to them regardless of other actions.
- The ranking problem concerns generating a partial or complete preferred pre-order of alternatives (Pδ).

Multi-criteria decision-making process

In practice, a multi-criteria decision-making problem is embedded in a wider process of problem structuring and resolution [19]. Figure 1 shows the main stages.

As illustrated in Fig. 1, each step in the decision-making process is an iterative and recurring step. In phase 4, decision stakeholders should perform several executions of the chosen MCDM method to check how robust the results are and study the degree of sensitivity of the result to the variation of the parameters. Indeed, robustness analysis

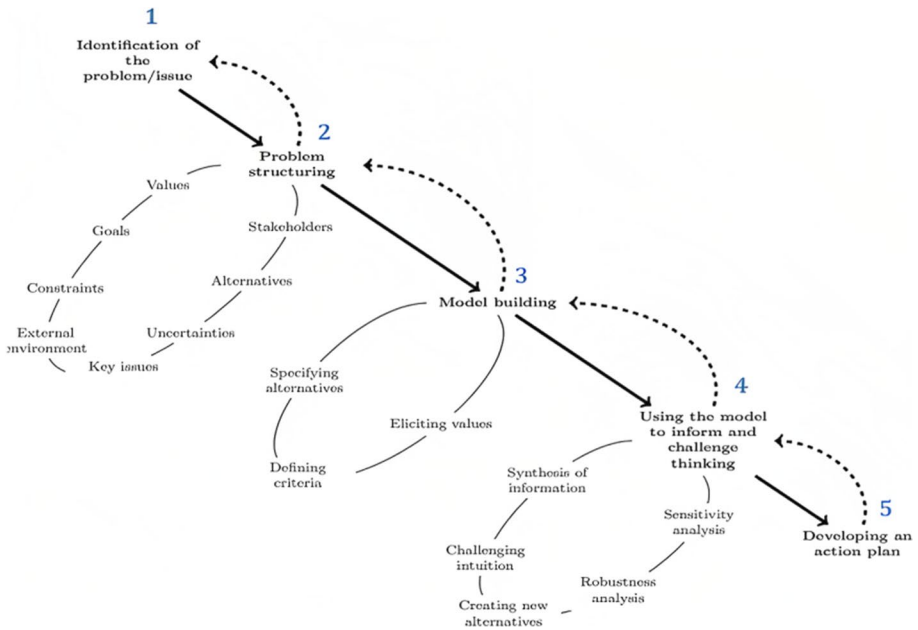

Fig. 1 Multi-criteria decision-making process [19]

Table 2 Decision matrix for the illustrative example

	Criteria				
M=	Weights	g_1	g_2	g_3	g_4
		1	1	1	1
		Max	Max	Max	Max
	A_1	4	4	5	3
	A2	3	3	4	2
	A3	5	4	2	2
	A4	1	2	3	1

We emphasize that the weights of the criteria are equal to 1, and all the criteria should be maximized

Table 3 Final ranking for the illustrative example

Alternative	**Rank**
Alternative_1	1
Alternative 3	2
Alternative_2	3
Alternative_4	4

consists of changing the input parameter values in order to observe what happens to the output results. The purpose of the sensitivity analysis is to find the interval $[V_{min}, V_{max}]$ in which the variation of the values of the parameters, such as the weight of each criterion, will not affect the final result [20, 21]. The principle consists of varying the model parameters to study the solution's stability (Tables 2 and 3).

Robustness and sensitivity analysis are crucial in a multi-criteria decision support process. The main disadvantage of these analyses is that they are expensive in computation time and require considerable effort on the part of those involved. Thus, it becomes crucial to obtain fast responses from the decision system support during this stage.

On the other side, MCDM problems have recently received attention from artificial intelligence, machine learning, and data mining communities [5, 9, 22]. In [6], authors propose a new model, based upon the concept of TOPSIS, to automatically classify credit score data into groups of high or low expected repayment. Several recent works have focused on consolidating data mining methods with MCDM methods to give DM tools to tackle decision-making problems in a big data context, such as supplier selection and personnel assessment [23, 24].

In the case of applying the TOPSIS method, the work presented in this article proposes using the MapReduce paradigm to reduce the execution time of each iteration and thus generate results within a reasonable time and support stakeholders in the decision-making process.

TOPSIS: definition and algorithm

The TOPSIS method and its stages

The technique for Order of Preference by Similarity to Ideal Solution (TOPSIS) method [25] is a robust approach to solving multi-criteria decision-making problems. First developed by Hwang and Yoon [25]. It is used to calculate the alternatives ratings. The

Positive Ideal Solution (PIS) and the Negative Ideal Solution (NIS) are two major notions used to find the optimal alternative. The positive ideal maximizes the profit criterion and minimizes cost, while the negative maximizes the cost criterion and minimizes profit. TOPSIS seeks to select the alternative with the shortest distance to the positive ideal solution and the longest to the negative one.

Due to its effectiveness and potentiality, TOPSIS has emerged as a popular MCDM method with a wide application domain [2]. It is exploited in several fields, such as Supply Chain Management, manufacturing systems, business and Marketing Management, Health, Human Resources Management, and Other topics.

Recently, many scientists have proposed a combination of business analytics (descriptive, predictive, and prescriptive) with the TOPSIS method to provide robust data-driven tools [26]. TOPSIS method was used as a data mining model, and in [6], it was used as a classification model for credit scoring. In [27], the authors proposed a Social Media Analytics model based on TOPSIS that works better than the standard methods.

TOPSIS consists of the following steps:

Step1: Construction of the decision matrix composed of n alternatives and m criteria.

Step2: Construction of the normalized decision matrix.

To make the preferences on all the criteria homogeneous and comparable and to transform all the criteria into criteria to be maximized, there are four normalization procedures in the literature which are the most commonly used [28]. We present their calculation formulas (1–1), (1–2), (1–3) and (1–4) below.

(i) Normalization procedure N1: by the maximum value of the scores

$$n_{ij} = \begin{cases} \dfrac{a_{ij}}{\max\limits_{k\in\{1,\ldots,n\}} a_{kj}} \; if\; j \in J^{+} \\ \\ 1 - \dfrac{a_{ij}}{\max\limits_{k\in\{1,\ldots,n\}} a_{kj}} \; if\; j \in J^{-} \end{cases} \tag{1-1}$$

(ii) Normalization procedure N2: by the ratio between the difference of the scores and the minimum value of the scores, and the difference between the maximum value and the minimum value of the scores

$$n_{ij} = \begin{cases} \dfrac{a_{ij} - \min\limits_{k\in\{1,\ldots,n\}} a_{kj}}{\max\limits_{k\in\{1,\ldots,n\}} a_{kj} - \min\limits_{k\in\{1,\ldots,n\}} a_{kj}} \; if\; j \in J^{+} \\ \\ 1 - \dfrac{\max\limits_{k\in\{1,\ldots,n\}} a_{kj} - a_{ij}}{\max\limits_{k\in\{1,\ldots,n\}} a_{kj} - \min\limits_{k\in\{1,\ldots,n\}} a_{kj}} \; if\; j \in J^{-} \end{cases} \tag{1-2}$$

(iii) Normalization procedure N3: by the sum of the scores

$$n_{ij} = \begin{cases} \dfrac{a_{ij}}{\sum_{k=1}^{n} a_{kj}} if\ j \in J^{+} \\ \\ 1 - \dfrac{a_{ij}}{\sum_{k=1}^{n} a_{kj}} if\ j \in J^{-} \end{cases} \tag{1-3}$$

(iv) Normalization procedure N4: by the square root of the sum of the squares of the scores.

$$n_{ij} = \begin{cases} \dfrac{a_{ij}}{\sqrt{\sum_{k=1}^{n} a_{kj}^{2}}} if\ j \in J^{+} \\ \\ 1 - \dfrac{a_{ij}}{\sqrt{\sum_{k=1}^{n} a_{kj}^{2}}} if\ j \in J^{-} \end{cases} \tag{1-4}$$

Where for the four normalization procedures:

- a_{ij} is the evaluation of the criterion j for the alternative a_i
- J^{+} is the subset of criteria to be maximized
- J^{-} is the subset of criteria to be minimized
- n_{ij} is the element of the normalized matrix, which varies between 0 and 1

In this work, we opted for the N4 normalization procedure used in the original version of the TOPSIS method [25]. In other more recent versions of the TOPSIS method, it is proposed to follow up with more than one normalization procedure, as in the TOPSIS-2N method [28] [29–31] where it is proposed to combine with the two normalization procedures N2 and N4. All obtained results in this study remain open and easily generalized to all standardization procedures.

Step3: Construction of the weighted normalized decision matrix

$$v_{ij} = \frac{n_{ij} \times w_j}{\sum_{k=1}^{m} w_k} \forall i, j \tag{2}$$

where w_j is the weight of importance assigned to the criteria j.

Step4: Determination of the positive ideal solution PIS and the negative ideal solution NIS

$$PIS = \left\{ \left(\max_i \left(v_{ij} | j \in j^{+} \right) \right), \left(\min_i \left(v_{ij} | j \in j^{-} \right) \right) \right\} = \left\{ v_1^{+} v_2^{+}, \ldots \right\} \tag{3}$$

$$NIS = \left\{ \left(\max_i \left(v_{ij} | j \in j^{-} \right) \right), \left(\min_i \left(v_{ij} | j \in j^{+} \right) \right) \right\} = \left\{ v_1^{-} v_2^{-}, \ldots \right\} \tag{4}$$

where:

- J^{+}: associated with the criteria having a positive impact to be maximized (benefit)

- J^-: associated with the criteria having a negative impact to be minimized (cost)

Step5: Calculation of L^2-distance between the target alternative i and the best/worst condition

$$d_i^+ = \sqrt{\sum_{j=1}^{m} \left(v_{ij} - v_j^+\right)^2 \forall j} \tag{5}$$

$$d_i^- = \sqrt{\sum_{j=1}^{m} \left(v_{ij} - v_j^-\right)^2 \forall j} \tag{6}$$

Step6: Calculate the similarity to the worst condition NIS

$$S_i = \frac{d_i^-}{d_i^- + d_i^+} \text{ for } 1 \leq \text{i } \leq n \tag{7}$$

Step7: Rank the alternatives according to S_i.

Algorithm 1 summarizes the steps of the basic TOPSIS method.

Algorithm 1: Sequential algorithm of the TOPSIS method
Input: M(a_{ij}) 1≤i≤n and 1≤j≤m : Decision matrix with the set of alternatives and their criteria WV: Criteria weights vector **Output:** L: List of alternatives sorted according to their relevance
Begin *N ← normalization(M) // see equation 1-4* *W ← weighted(N,WV) // see equation 2* *PIS ← ideal_solution(W) // see equation 3* *NIS ← worst_ideal_solution(W) // see equation 4* *D^+ ← distance(W,PIS) // see equation 5* *D^- ← distance(W,NIS) // see equation 6* *S ← similarity_worst(D^+,D^-) // see equation 7* *L ← rank the alternatives based on S_i values* **End**

The complexity of the sequential algorithm of the TOPSIS method

The term used in algorithms to qualify the performance of an algorithm is the complexity, and more precisely, the time complexity and Space complexity. Time complexity is measured by the number of computing stages required to execute an algorithm as a function of data input size n. At the same time, the space complexity is measured by the memory used by the data structure contained in the algorithm as a function of n [32, 33].

Time complexity is the most widely used metric to assess the performance of an algorithm. Time is an essential factor, especially in a decision-making process. An algorithm is classified based on the time it takes to complete compared to the input size. There is a range of varieties. Some algorithms complete in a linear time relative

to the input size; others complete in a time that is quadratic $O(n^2)$; others complete in an exponential or worse time; and others stop [32].

With a quadratic algorithm, the time is increased proportionally to the square of the number of data. TOPSIS method has a higher time complexity of the order $O(n^2)$ [34]. This quadratic complexity, therefore, presents a limitation for decision-making problems with large data sizes. A parallel programming model, such as MapReduce, is a potential solution to overcome this limitation.

MapReduce paradigm

The availability of robust computing infrastructures and the need to process a gigantic volume of data have imposed the use of sophisticated and rapid programming models. One of the proven effective parallel computing models is MapReduce [35]. The framework of MapReduce is very convenient for distributed computing, which can abstract away from many difficulties in parallelizing data management operations through a cluster of machines [35]. Many MapReduce-based parallel computing platforms have been developed for the analysis of large-scale data sets, such as Hadoop [37, 38], Spark [39], etc.

The MapReduce model was proposed by two engineers at Google, who observed that many massively parallel processing operations, implemented for the needs of their search engine, followed an identical parallelization strategy. From these observations was born the MapReduce programming model, first described in 2004 [16]. The Map inspires his processing abstraction and Reduce primitives present in many functional languages like Lisp. MapReduce jobs processing includes two important phases: Map and Reduce. Each phase uses key-value pairs. Programmers specify what to do in the two functions, map() and reduce(). All values with the same key are sent to the same reducer. The execution framework handles everything else. Figure 2 illustrates the MapReduce workflow [35].

Spark via Hadoop

A MapReduce program must be supported by a dedicated software infrastructure that allows the MapReduce scheme to be executed in a massively distributed manner on a cluster of machines while guaranteeing the challenges of distributed computing:

- Optimization of disk and network transfers by limiting data movement,
- Scalability and fault tolerance

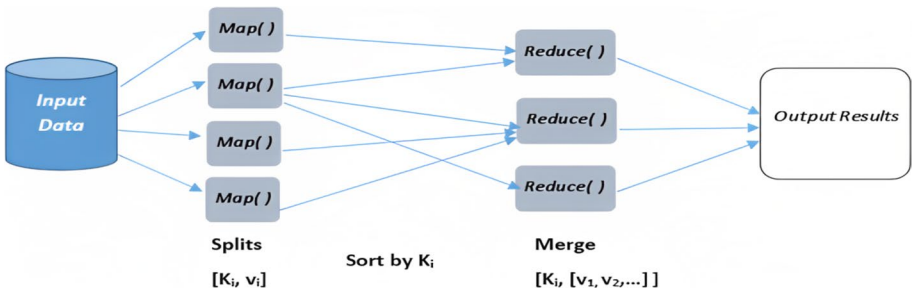

Fig. 2 MapReduce workflow

The Hadoop MapReduce platform has two major drawbacks [35, 36]:

1. After a map or reduce operation, the result should be written to disk. This data written to disk allows mappers and reducers to communicate with each other. It is also writing to disk that provides fault tolerance: if a map or reduce operation fails, data is retrieved from the disk. However, these writes and reads are very time-consuming.
2. The set of expressions composed exclusively of map and reduce operations is very limited and not very expressive. In other words, it is difficult to express complex operations using only this set of two operations.

Apache Spark is an alternative to Hadoop MapReduce for distributed computing that aims to solve these problems. The fundamental difference between Hadoop MapReduce and Spark is that Spark writes data to RAM, not disk. Thus, we achieve a performance increase by minimizing disk usage with Spark, especially for the programs with many iterative calculations sharing the same data [37].

This has several important consequences on the processing speed of calculations. However, storing the data of the intermediate calculations in RAM poses a challenge to overcome to guarantee fault tolerance. When a machine becomes unavailable, the data stored in RAM becomes inaccessible. To solve this problem, Spark distributes the calculations as a graph. The state of a node can be reconstructed from its neighboring nodes (Fig. 3).

Spark cluster architecture

A Spark cluster is composed of:

- One or more workers: each worker instantiates an executor responsible for executing the various calculation tasks. Each worker has multiple executors. This allows for running multiple Spark applications on a single machine at the same time.
- A driver: in charge of distributing the tasks to the different executors. This is the driver that executes the main method of our applications.
- A cluster manager: responsible for instantiating the different workers.

The figure below (Figure 3) gives an overview of a Spark cluster architecture [38].

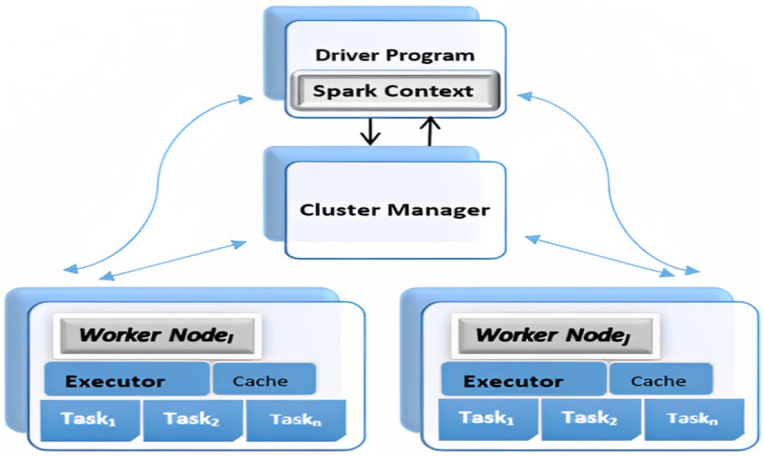

Fig. 3 Spark cluster architecture

The support for parallelism in the Spark architecture is incorporated through the Resilient Distributed Dataset (RDD) concept. RDD is defined as an immutable distributed collection of objects [39]. Each dataset in RDD is divided into logical partitions, which may be computed on different cluster nodes. And it can support two different kinds of operations: Transformations, which create a new RDD from the existing RDD, and actions that return final results to the driver program or the external storage.

Proposed approach: distributed TOPSIS approach and MapReduce calculation

The design of a distributed parallel TOPSIS algorithm is motivated by, firstly, the need to reduce computing time for large-scale datasets. Secondly, taking advantage of technological advances in computers.

TOPSIS method has a high time complexity of order $O(n^2)$. For a large-scale decision matrix, it will slow down the overall DM process. We have adopted a parallel algorithm based on the MapReduce paradigm to solve this problem. The calculations are distributed over several nodes.

To reformulate the operations of the TOPSIS algorithm into Map() and Reduce() functions, the question is how to define the input form of the data as: (key, value). And how to aggregate the values according to the calculations required at each stage. To answer these two questions, as shown in the illustrative numerical example of algorithm 2 above, we use the most commonly used transformers below.

First, the decision matrix data and criteria weights are stored on the RDDs created by the sc.parallelize() methods of the PySpark API. All the transformations applied to these RDDs are executed in a distributed and parallel way and produce as a result other RDDs [40].

- map() is a transformation operation that applies the transformation on every element of RDD and returns a new RDD. The output of map transformations would always have the same number of records as the input.
- flatmap() is a transformation that flattens the RDD after applying the function on every element and returns a new RDD. The returned RDD could return more rows. It is also referred to as a one-to-many transformation function.
- reduceByKey() is a transformation used to aggregate the values for each key using an associative and commutative reduce function like min and max.
- sortByKey() is a transformation used to sort the result by key.
- collect() is an action operation that is used to retrieve all the elements of the RDD dataset, from all nodes to the master node.

Distributed TOPSIS algorithm

The implementation of TOPSIS algorithms starts by creating a spark context using the SparkContext() function after loading data from the input dataset. Data is partitioned into several fragments. Each Spark's executor node receives a different data partition

in the form of RDD. The pseudocode of the TOPSIS algorithm with the MapReduce model is given in Algorithm 2.

The SparkContext.parallelize() method is used to create parallelized collection of RDD from a list of collections which can be distributed over a cluster for parallel processing.

Algorithm 2: Distributed TOPSIS algorithm with Spark and MapReduce implementation

Input:
M(a_{ij}) $1 \le i \le n$ and $1 \le j \le m$: Decision matrix with the set of alternatives and their criteria
WV: Criteria weights vector
Output:
L: List of alternatives sorted according to their relevance

Begin
Initialize SparkContext
data ← M //reading the decision matrix
W ← WV // reading the weights of the criteria
rdd ← sc.parallelize(data) // Create RDD for the decision matrix
rdd1 ← map(rdd) //Used with: x.split(",") for splitting each row in different values with "," separator
rdd2 ← flatmap(rdd1) // Used with : float(x[i]*x[i]) for i in list[0:criteria_number] for calculate the squares
reduce1 ← reduceByKey(rdd2, lambda x,y: x+y) // Calculate the sum of squares
N← map(rdd1,reduce1) // Normalization matrix
Weights ← sc.parallelize(W).collect( ) // Create RDD for the weights of the criteria
Weighted_matrix ← map(N, Weights) // Normalized matrix weighting
flat_weighted_matrix ← flatmap(weighted_matrix) // Normalized matrix weighting with key
PIS ← reduceByKey(flat_weighted_matrix, max( )) // Calculation of the positive ideal solution
NIS ← reduceByKey(flat_weighted_matrix, min( )) // Calculation of the negative ideal solution
positive_distance← map(Weighted_matrix, PIS) // Calculation of positive distance D+
negative_distance← map(Weighted_matrix, NIS) // Calculation of negative distance D-
D_simularity ← map(positive_distance, negative_distance).SortByKey( ) // Calculation of the similarity and
L← Return sorted alternatives key // Sort alternatives in descending order of similarity
End

Numerical illustration of the calculations of algorithm 2

To illustrate the step-by-step execution of Algorithm 2, whose source code is provided as an attachment to this article (via the link: related files), we present an example MCDM problem with four alternatives and four criteria assumed to be maximized (see Table 2).

To understand how the data will be aggregated to form the final ranking of the alternatives, all the calculation steps with the functions map(), flatmap() and reduceByKey() are explained numerically. See Figs. 4, 5, 6, 7 and 8.

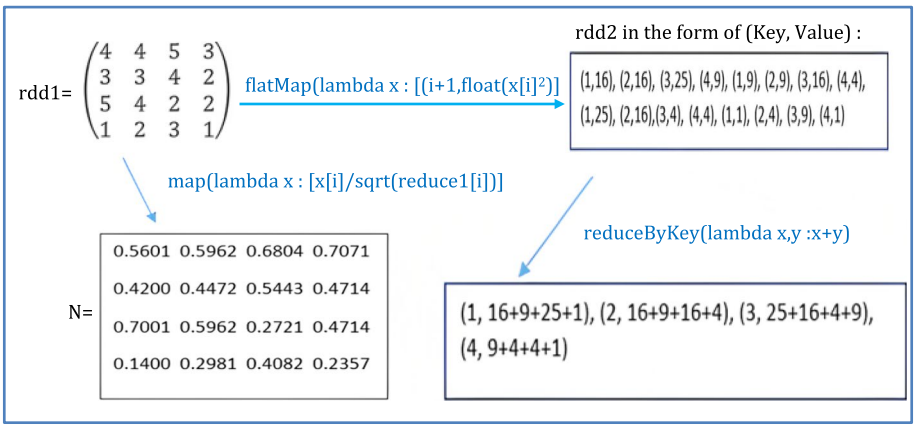

Fig. 4 Matrix normalization

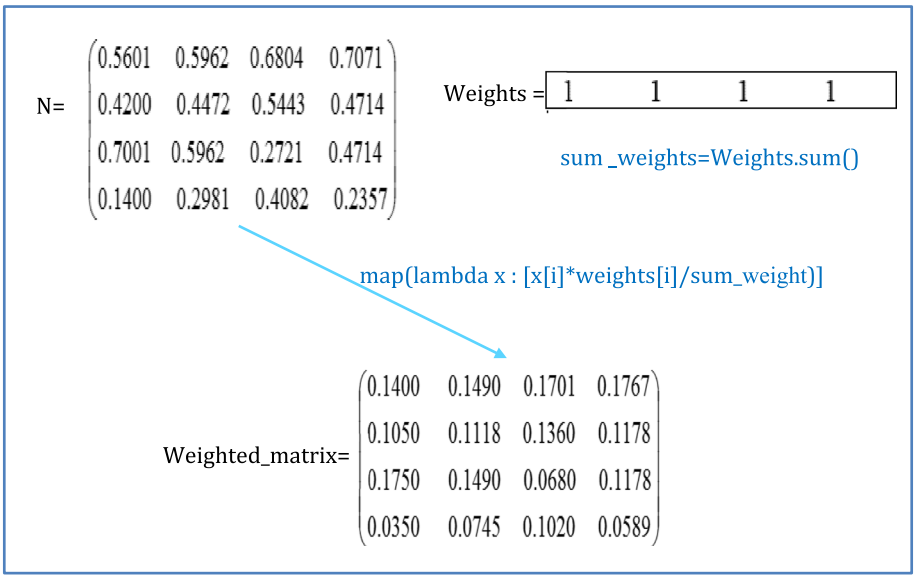

Fig. 5 Calculation of the weighted matrix

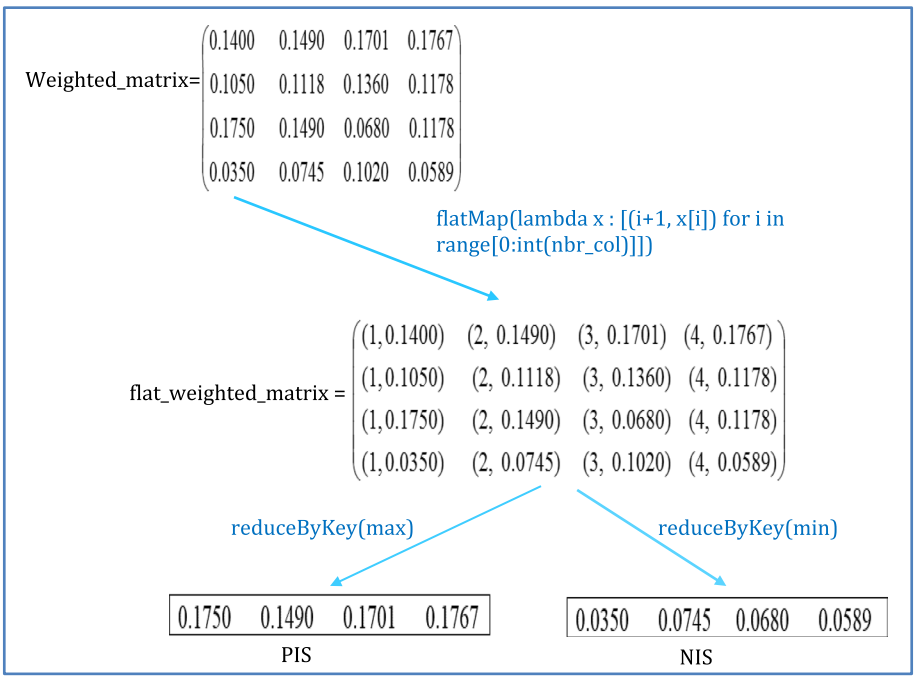

Fig. 6 Calculation of PIS and NIS

For the decision matrix normalization step, the figure shows how the data was distributed as a key value with the flatMap() function and how the intermediate results were aggregated with the reduceByKey() function. See Figs. 4 and 5 below.

To compute the ideal solution, we transform *the weighted normalization of the matrix* as a set of pairs (key, value). For example, for the alternative A1: we consider

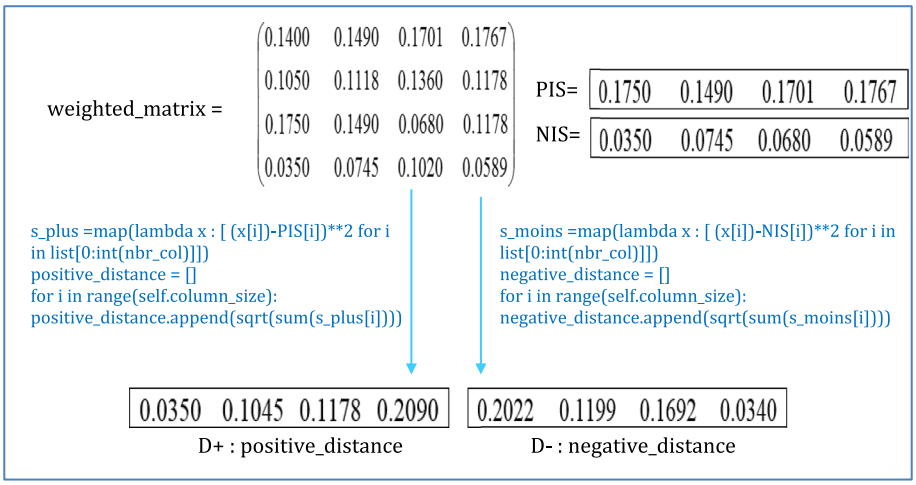

Fig. 7 Calculation of positive_distance and negative_distance

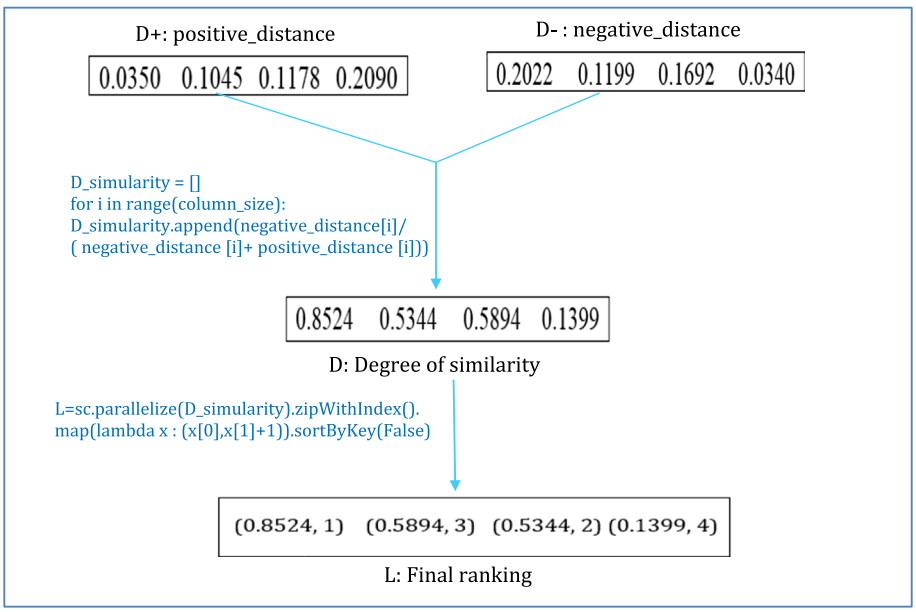

Fig. 8 Calculation of the degree of similarity and the final ranking

the pairs (key, value): (1, a_{11}), (2, a_{12}),...,(m, a_{1m}) where a_{1j} represents the performance of the alternative A1 on the jth criterion (from the weighted matrix). The reduce operation then retrieves the maximum value (function: reduceByKey(max)) among the values associated with a given key.

The following figure illustrates the same principle used to calculate the positive ideal solution (PIS) and negative ideal solution (NIS).

In conclusion for this example illustration of the application of algorithm 2, the ranking in descending order of the four alternatives based on the decision matrix and the weighting provided as input is as follows (see Table 3):

Experimentation and discussion of results

Datasets selected for experimentation

To verify the effectiveness of our approach, we performed our experiments on one randomly generated dataset and two real datasets (all deployed data files are attached to this paper via the link: related files).

1. Generated data set (https://generatedata.com/): To analyze the variation of the execution time according to the number of alternatives, we generated several databases. Each time, we increase the number of lines (from 100 lines up to 1 million) and the number of criteria (10 to 40 criteria: a number rarely reached in an MCDM problem).
2. Mobile price data set (https://www.kaggle.com/datasets/iabhishekofficial/mobile-price-classification): this database contains 2000 rows and 21 numerical columns. Each line represents the characteristics of a phone (like internal memory in GB, clock speed, number of cores, weight, pixel resolution price, dual sim etc.). The choice of a phone presents an MCDM problem. The choice or order of preference of phones depends on the importance (weight) given to each criterion. This importance differs from one person to another.
3. Credit Card Clients Dataset (https://www.kaggle.com/datasets/uciml/default-of-credit-card-clients-dataset?select=UCI_Credit_Card): This dataset contains information on default payments, credit data, history of payment, and bill statements of credit card clients. This resource presents information relating to bank customers. It allows sorting according to several criteria such as level of education, age, repayment status, amount of bill statement, amount of previous payment, default payment etc. It has 30 000 rows and 23 criteria.

Experiment environment

We have run the distributed algorithm proposed in GCP's Dataproc, a managed service for running Apache Hadoop and Spark jobs in Google Cloud. We have configured two different cluster configurations:

- In the first configuration, we used a cluster with a master node and two workers (the master node has an n1-standard-2 system with 2 CPUs with 7.5 GB memory each. Worker nodes have the same configuration).
- In the second configuration, we used a cluster with a master node and three worker nodes (the same characteristics as the first master node).

We have also run the sequential algorithm of TOPSIS in GCP using an n1-standard-1 system (1 vCPU, 3.75 GB memory).

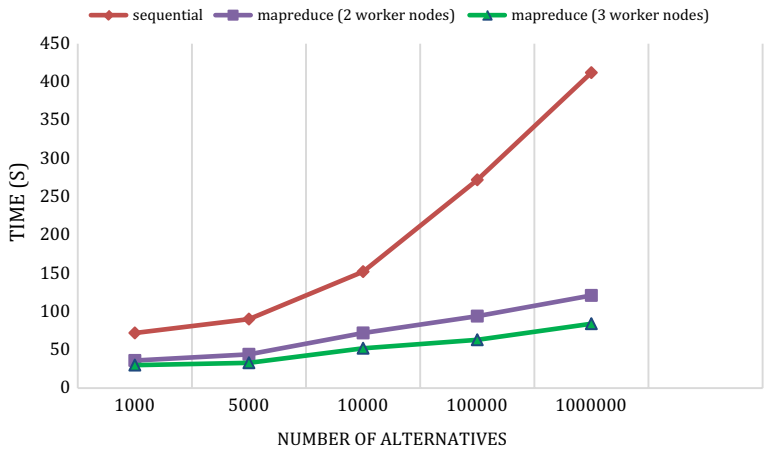

Fig. 9 TOPSIS Sequential and Distributed TOPSIS Computation Time

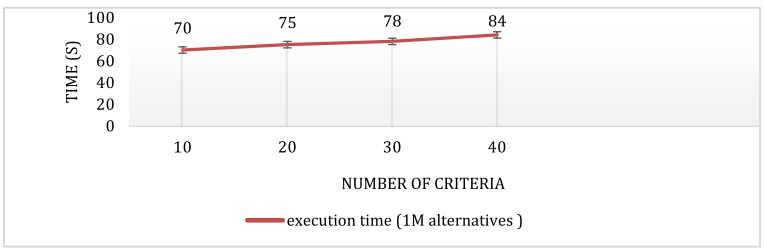

Fig. 10 Execution time according to the number of criteria (cluster with three worker nodes)

Discussion of results

As shown in the proposed algorithm 2, we used the MapReduce paradigm to rank a set of alternatives each described by a family of criteria. The objective is to process the alternatives at several levels in parallel: at the first level, which consists in normalizing the performance matrix into a weighted matrix, at the second level, which is used to calculate the two reference solutions PIS and NIS, at the third level for calculate the euclidean distances between the alternatives and the solutions PIS and NIS, and finally at the fourth level to calculate the degree of similarity of each alternative and classify the alternatives according to these degrees of similarity. However, in the original version of TOPSIS, as illustrated in Algorithm 1, these fourth calculation levels were performed sequentially. Thus, we have deployed the famous methods map(), flatmap(), and reduceByKey() [40] to divert the sequential processing, which undoubtedly requires a much higher execution time than the parallel version.

In this section, we examine all the results obtained by the different tests carried out.

Figure 9 concerns results regarding generated data sets (results regarding the other data sets lead to similar conclusions). We use the size of the dataset as a metric to evaluate the execution time. The number of criteria was also considered as a metric (curve in Fig. 10). From the observed experimental results in Fig. 9, it is clear that

executing time grows with the number of alternatives. We notice that the execution time in the cluster with three worker nodes is much lower than in the sequential algorithm.

The computation has been performed with MapReduce implementation over the Dataproc cluster for all different-size data sets. The sequential algorithm has taken approximately 412 s (≈7 min) for 1 million alternatives, while Dataproc clusters with two worker nodes and three worker nodes have taken approximately 121 and 84 s, respectively. Thus, we can improve the time efficiency by deploying more worker nodes.

The curve in Fig. 10 shows the variation in the computation time according to the number of criteria. It concerns a data set with 1 M alternatives and a cluster with three worker nodes. The number of criteria does not have much influence knowing that the number of criteria in a real decision-making problem remains limited. Tests carried out confirmed what we mentioned before. The number of alternatives contributes more to increasing the execution time than the number of criteria.

The experimentation carried out in this present research work proves that the proposed Distributed-TOPSIS approach has largely overcome the problem of the execution time, which remains very high for sets containing a considerable number of alternatives. This execution time is obviously linked to the high temporal complexity imposed by the sequential version of the TOPSIS method. Thus, the Distributed-TOPSIS approach is a notable improvement of the basic version; indeed, the execution time is reduced from seven minutes to less than two minutes for a set of up to one million alternatives.

This new approach can therefore be adopted for any multi-criteria decision problem in a big data context without having to filter specific alternatives to reduce the size of the set of alternatives to be processed. In addition, this same version can be deployed to help decision-makers find the best and most stable solution by the robustness analysis operation, which recommends running the method several times with different parameters, such as the criteria weights and evaluations of some alternatives.

For the robustness of this research work, we make available to readers, as attachments to this paper, the complete PySpark program of the proposed parallel distributed approach of the TOPSIS method and the data files tested (via the link: related files).

Conclusions and further research

This work presents a MapReduce implementation of TOPSIS, one of the most robust MCDM methods. Using this paradigm, a computing response time that could be disappointing for decision-makers may be reduced. This improvement allows for dealing with large data sets and the uncertainties generated during several iterative executions of TOPSIS in the robustness and sensitivity analysis phase.

For experiments with the distributed parallel algorithm of the TOPSIS method, we took advantage of the distributed servers infrastructure of the Google Cloud Platform with a trial account. However, decision-makers for a practical application of the proposed approach have many other choices of cloud computing platforms, such as Amazon Web Services and Microsoft Azure, with the possibility of using many worker nodes to improve execution time.

This research opens up multiple prospective for future extensions. The idea of the parallel distributed implementation based on the MapReduce paradigm proposed in this present work can be judiciously generalized to other MCDM methods. We aim to design and develop an integrated Big-Data model for MCDM methods like MLlib Spark that comes with a library containing common machine learning (ML) functionality. The authors are presently working in this direction.

In the age of the data explosion, the use of new advances in computing is becoming a common and inevitable choice. In the future, we are interested in using quantum programming to speed up the calculation time. Quantum programming is a breakthrough technology that could limit what we can compute, but its future is still fuzzy.

Acknowledgements
The authors thank the president of the University Sidi Mohmed Ben Abdellah, the dean of the Faculty of Science and Technology and the director of the Laboratory of Intelligent Systems and Applications for their continuous encouragement and support.

Author contributions
All mentioned authors contribute to the elaboration of the paper. All authors read and approved the final manuscript.

Authors' information
Loubna Lamrini is a computer engineer and professor in the Department of Computer Engineering, Faculty of Sciences and Techniques of the University Sidi Mohamed Ben Abdellah, Fez Morocco. She obtained her degree in computer engineering from ENIM, Rabat, Morocco. Loubna Lamrini is actively engaged in research on various aspects of information technology ranging from multi-criteria decision analysis methods and high-performance computing. Mohammed Chaouki ABOUNAIMA received a Ph.D. in Computer Science from the Mohammed V University, Rabat, Morocco, in 1997. He is a professor at the Department of Computer Engineering, Faculty of Science and Technology Fez Morocco. He is a member of the Laboratory of Intelligent Systems & Applications. His research interests lie in data analysis, data mining, machine learning, and multi-criteria decision-making. Mohammed Talibi Alaoui obtained his PH Degree in automatic and information processing from Ibn Tofail University, Kenitra, Morocco. He received a Habilitation degree in 2014 from the Mohammed First University, Oujda (Morocco). He is currently a professor at the Sidi Mohammed Ben Abdellah University of Fes (Morocco). He teaches computer science. His field of research interest is in image processing, computer vision, and their applications to medical imaging, quality control, and parallel computing.

Funding
Not applicable.

Availability of data and materials
yspark program file: distopsis. Data files: actions1M.csv, mobile_choice.csv, clients_data.csv (via the link: related files)

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 21 November 2022 Accepted: 22 May 2023
Published online: 09 June 2023

References

1. Roy B. Decision-aid and decision-making. Eur J Oper Res. 1990;45(2–3):324–31. https://doi.org/10.1016/0377-2217(90)90196-I.
2. Behzadian M, Khanmohammadi Otaghsara S, Yazdani M, Ignatius J. A state-of-the-art survey of TOPSIS applications. Expert Syst Appl. 2012;39(17):13051–69. https://doi.org/10.1016/j.eswa.2012.05.056.
3. Yadav V, Karmakar S, Kalbar PP, Dikshit AK. PyTOPS: a python based tool for TOPSIS. SoftwareX. 2019;9:217–22. https://doi.org/10.1016/j.softx.2019.02.004.
4. Jablonsky J. MS Excel based software support tools for decision problems with multiple criteria. Procedia Econ Financ. 2014;12(March):251–8. https://doi.org/10.1016/s2212-5671(14)00342-6.

5. Selcuk A, Selcuk H, Delen D. Technological forecasting & social change the use of multi-criteria decision-making methods in business analytics: a comprehensive literature review. Technol Forecast Soc Chang. 2022;174:121193. https://doi.org/10.1016/j.techfore.2021.121193.
6. wu D, Olson DL. A TOPSIS data mining demonstration and application to credit scoring. Int J Data Warehous Min. 2006;2(3):16–26. https://doi.org/10.4018/jdwm.2006070102.
7. Mahdiraji HA, Zavadskas EK, Kamardi AA. Marketing strategies evaluation based on big data analysis : a CLUSTERING-MCDM approach. Economic Res Ekonomska Istraživanja. 2019. https://doi.org/10.1080/1331677X.2019.1658534.
8. A Azadnia, Hossein P, Ghadimi M, Molani-Aghdam, "Title A hybrid model for estimating customer lifetime value." 2011. of data mining and MCDM methods
9. Yang M, Nazir S, Xu Q, Ali S, Uddin MI. Deep learning algorithms and multicriteria decision-making used in big data: a systematic literature review. Complexity. 2020. https://doi.org/10.1155/2020/2836064.
10. Zavadskas EK, Turskis Z, Kildiene S. State of art surveys of overviews on MCDM/MADM methods. Technol Econ Dev Econ. 2014;20(1):165–79. https://doi.org/10.3846/20294913.2014.892037.
11. Chen Y, Kilgour DM, Hipel KW. Screening in multiple criteria decision analysis. Decis Support Syst. 2008;45(2):278–90. https://doi.org/10.1016/j.dss.2007.12.017.
12. Lamrini L, Abounaima MC, Talibi M, Alaoui M, Ouzarf FZ, Mazouri EL. MCDM filter with Pareto parallel implementation in shared memory environment (SOIC-1216). Stat Optim Inf Comput. 2022. https://doi.org/10.19139/soic-2310-5070-1216.
13. Lamrini L, Abounaima MC, Talibi Alaoui M, El Mazouri FZ, El Makhfi N, Ouzarf M, ". A filtering approach used in a massive data context to reduce the set of choices in a multicriteria decision aid process: Pareto solutions. Int Conf Electron Control Optim Computer Sci. 2020. https://doi.org/10.1109/ICECOCS50124.2020.9314445.
14. Guillet BD, Mattila A, Gao L. The effects of choice set size and information filtering mechanisms on online hotel booking. Int J Hospitality Manag. 2019. https://doi.org/10.1016/j.ijhm.2019.102379.
15. Kahneman D, Snell J. Predicting a changing taste: do people know what they will like? J Behav Decis Mak. 1992;5(3):187–200.
16. Zhang J, Zhang X, Zhang W. Microseismic search engine. Soc Explor Geophys Int Expo 83rd Annu Meet SEG 2013 Expand Geophys Front. 2013. https://doi.org/10.1190/segam2013-1277.1.
17. Chernev A, Böckenholt U, Goodman J. Choice overload: a conceptual review and meta-analysis. J Consum Psychol. 2012;25(2):333–58. https://doi.org/10.1016/j.jcps.2014.08.002.
18. B. Roy and D. Bouyssou, "Aide multicritere à la décision: méthodes et cas. Production et techniques quantitatives appliquées à la gestion," *Econ. Paris, Fr.*, 1993.
19. Belton V, Stewart T. Problem structuring and multiple criteria decision analysis. Int Ser Oper Res Manag Sci. 2010;142:209–39. https://doi.org/10.1007/978-1-4419-5904-1_8.
20. DG-M. criteria decision making and undefined 2012, "Sensitivity and robustness analysis of solutions obtained in the European projects' ranking process," *mcdm.ue.katowice.pl*.. https://mcdm.ue.katowice.pl/files/mcdm12.pdf#page=86. Accessed 26 Apr 2023.
21. Song JY, Chung E. Robustness, uncertainty and sensitivity analyses of the TOPSIS method for quantitative climate change vulnerability : a case study of flood damage. Water Resour Manag. 2016. https://doi.org/10.1007/s11269-016-1451-2.
22. El Mazouri FZ, Abounaima MC, Zenkouar K. Data mining combined to the multicriteria decision analysis for the improvement of road safety: case of France. J Big Data. 2019. https://doi.org/10.1186/S40537-018-0165-0.
23. Ijadi Maghsoodi A, Kavian A, Khalilzadeh M, Brauers WKM. CLUS-MCDA: a novel framework based on cluster analysis and multiple criteria decision theory in a supplier selection problem. Comput Ind Eng. 2018. https://doi.org/10.1016/j.cie.2018.03.011.
24. Ijadi Maghsoodi A, Riahi D, Herrera-Viedma E, Zavadskas EK. An integrated parallel big data decision support tool using the W-CLUS-MCDA: a multi-scenario personnel assessment. Knowledge-Based Syst. 2020. https://doi.org/10.1016/j.knosys.2020.105749.
25. Hwang C-L, Yoon K. Methods for Multiple Attribute Decision Making. In: Hwang C-L, Yoon K, editors. Multiple Attribute Decision Making. Berlin: Springer Berlin Heidelberg; 1981.
26. Nilashi M, Mardani A, Liao H, Ahmadi H, Manaf AA, Almukadi W. A hybrid method with TOPSIS and machine learning techniques for sustainable development of green hotels considering online reviews. Sustainability. 2019. https://doi.org/10.3390/su11216013.
27. Muruganantham A, Gandhi GM. Framework for social media analytics based on multi-criteria decision making (MCDM) model. Multimed Tools Appl. 2019. https://doi.org/10.1007/S11042-019-7470-2.
28. De Siqueira Silva MJ, et al. A comparative analysis of Multicriteria methods AHP-TOPSIS-2N, PROMETHEE-SAPEVO-M1 and SAPEVO-M: selection of a truck for transport of live cargo. Procedia Comput Sci. 2022. https://doi.org/10.1016/J.PROCS.2022.11.152.
29. De Souza LP, Gomes CFS, De Barros AP. "Implementation of New Hybrid AHP–TOPSIS-2N method in sorting and prioritizing of an it CAPEX project portfolio. Int J Info Technol Dec Mak. 2018. https://doi.org/10.1142/S0219622018500207.
30. Silvado MC, Gomes CFS, Souza RC. "TOPSIS-2NE's proposal. Int J Fuzzy Syst. 2020. https://doi.org/10.1007/S40815-020-00871-4/METRICS.
31. Silva MDC, Gomes CFS, Da Costa Junior CL. A hybrid Multicriteria methodology topsis-macbeth-2n applied in the ordering of technology transfer offices. Pesqui Oper. 2018. https://doi.org/10.1590/0101-7438.2018.038.03.0413.
32. B Bollig and I Wegener. 2005 complexity theory exploring the limits of efficient algorithms. 32(4)
33. Burgin M. Algorithmic complexity as a criterion of unsolvability. Theor Comput Sci. 2007;383(2–3):244–59. https://doi.org/10.1016/j.tcs.2007.04.011.
34. R. Wardoyo, "The complexity calculation for group decision making using TOPSIS algorithm," 2017, doi: https://doi.org/10.1063/1.4958502.
35. H. Jin, S. Ibrahim, L. Qi, H. Cao, S. Wu, and X. Shi. 2011 "The MapReduce Programming Model and Implementations," in *Cloud Computing: Principles and Paradigms*, John Wiley and Sons. 373–390.
36. Asghar H, Nazir B. Analysis and implementation of reactive fault tolerance techniques in Hadoop: a comparative study. J Supercomput. 2021;77(7):7184–210. https://doi.org/10.1007/s11227-020-03491-9.

37. Mavridis I, Karatza H. Performance evaluation of cloud-based log file analysis with apache Hadoop and apache spark. J Syst Softw. 2017;125:133–51. https://doi.org/10.1016/j.jss.2016.11.037.
38. "Cluster Mode Overview—Spark 3.3.0 Documentation." https://spark.apache.org/docs/3.3.0/. Accessed 26 Apr 2023.
39. H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark LIGHTNING-FAST DATA ANALYSIS*. O'Reilly Media, 2015.
40. "PySpark Tutorial For Beginners|Python Examples - Spark By {Examples}." https://sparkbyexamples.com/pyspark-tutorial/. Accessed 26 Apr 2023.

Publisher's Note