GFEA: Leader Election Algorithm for Choosing a Group Decision Support System Facilitator

Mohammedi Taieb Sabir[1], Laredj Mohamed Adnane[1]

[1]*Lab CSTL, Mostaganem University, 27000 Mostaganem, Algeria.*

Abstract

Group decision support systems (GDSSs) are computer-assisted collaborative work software that facilitates group meetings asynchronously and from different locations. Even so, collaborative work in GDSS demands coordination provided by a single controlling entity known as the GDSS facilitator. However, the problem of electing a GDSS Facilitator hasn't been treated enough in the literature, and it is often neglected. Despite that, the large number of responsibilities assigned to the facilitator makes his role crucial to the effectiveness of the group meeting. Thus, the authors focused on finding an appropriate approach for electing the facilitator. The similarities between the problematics of electing a GDSS facilitator and a distributed system leader led the authors to consider applying a distributed election algorithm for electing a GDSS facilitator. Nonetheless, current algorithms only consider computer criteria and lack a formal weighting method. Consequently, we proposed a new distributed election algorithm called GFEA (GDSS Facilitator Election Algorithm) that is designed to choose a facilitator within a GDSS. This algorithm selects a facilitator among a set of decision-makers based on multiple election criteria weighted using an objective weighting method called MEREC. A backup leader is reserved to replace the leader if he fails, and a new tie-breaking mechanism is proposed. Moreover, the initiator failure is handled. By adopting distributed system leader election principles, GFEA provides a robust solution for a decisive GDSS challenge.

Keywords*: Leader Election, Distributed Systems, GDSS, Facilitator, Multi-criteria.*

1. Introduction

A distributed system is a group of computers or mobile devices connected through a network. These devices work together, appearing as a single unit to users, to achieve a common goal and deliver a service [1]. These systems are employed in various fields, they are integrated into banking networks, smart container systems, smart plants, industry 4.0, IoT, smart cities, and many other application cases [2]. Coordinating the nodes within the system is the leader's responsibility. The system's leader is responsible for allocating resources, balancing the load on the different nodes, coordinating the consensus regarding replicated data, and handling deadlock situations [3].

GDSS (Group Decision Support System) is a DSS (Decision Support System) that's designed to be used by a group of DMs (Decision Makers), who communicate using a communication subsystem. It's a combination of a group of humans, hardware, and software. The GDSS enhances the group decision-making process of organizations [4]. The humans using the GDSS are grouped into two roles: the DMs and the facilitator [5]. It is necessary to have a human facilitator within a GDSS [4]. The traditional decision room is composed of multiple computers connected using a local network [6]. Nonetheless, it can also be extended to support connecting DMs who are geographically dispersed through the internet [7]. One of the DMs is handed an important role

Corresponding author: *Mohammedi Taieb Sabir(sabir.mohammeditaieb.etu@univ-mosta.dz)*
Received: *30 September 2024*; Revised: *29 March 2025*; Accepted: *23 April 2025*; Published: *15 May 2025*

called the facilitator role. The GDSS facilitator is responsible for various tasks. He walks the DMs through the meeting's agenda and starts the group conversation. Additionally, the facilitator can bring up new ideas if the DMs agree with that. Moreover, he can enhance the performance and effectiveness of the group. Furthermore, the facilitator helps the DMs use the technologies and do their tasks. On top of this, he has to clarify the results of the meeting [8].

Despite the importance and the significant number of roles assigned to the GDSS facilitator, there hasn't been enough attention regarding the selection of a GDSS facilitator. Consequently, this work is dedicated to designing a solution for this problematic. This problematic is formulated using the following questions: Among the set of DMs, how can we select one of them and assign him the role of GDSS facilitator? and how are DMs evaluated to find the most suitable DM for the facilitator role? Due to the similarities between the problematic of electing a leader in distributed systems and the problematic of electing a GDSS facilitator, the distributed leader election algorithm represents a potential solution to the problematic of this work. Despite that, the solution must satisfy five requirements to be suitable for the problematic at hand. These requirements are: Consideration of multiple election criteria, inclusion of human-related criteria, a formal weighting method, a backup leader, and the ability to handle the failure and recovery of both the leader and the election initiator.

Among the various election algorithms that have been suggested in the literature (see Literature Review), some works integrate some of the required features for electing a GDSS facilitator. But no algorithm satisfies all the requirements of this problematic. Additionally, no distributed election algorithm optimized for this problematic has been proposed. Moreover, the existing algorithms are optimized for distributed machines and not humans. The work in [8] proposes an election algorithm based on the load and failure rate of the nodes. While this algorithm involves multiple criteria, it doesn't consider any human-related criterion, plus it doesn't specify a formal weighting method.

In this paper, a new distributed election algorithm designed for the GDSS facilitator election is proposed. The proposed algorithm considers multiple election criteria, including human experience, while using a proper weighting method to indicate the importance of each criterion, thus influencing the final results. Furthermore, the algorithm elects not only the facilitator but also a backup leader. Moreover, the GFEA (GDSS Facilitator Election Algorithm) satisfies all three correctness properties and adds a new tie-breaking mechanism that considers the most important criteria instead of just relying on the UID (Unique Identifier). On top of that, the algorithm is fault-tolerant and considers the disconnection of DMs, the leader, and even the initiator. This algorithm can also be applied to machines by changing the election criteria according to the problem at hand. This algorithm allows us to solve the problematic of electing a GDSS facilitator formally and objectively, that doesn't involve subjective parameters. Thus, eliminating biases and human conflicts that could arise when using a subjective approach like multi-criteria decision-making methods [9]. The main contributions of the proposed work are as follows:

- Comparison between recent election algorithms designed for distributed systems, and spotting similarities between the problematic of electing a GDSS facilitator and the problematic of electing a leader for a distributed system. By exploiting these similarities, it is demonstrated that the distributed election algorithms can be effectively adapted to solve the problematic of electing a GDSS facilitator.
- Proposition of the GFEA algorithm optimized for the problematic of electing a GDSS facilitator based on human experience, security, and network performance criteria.
- Brief comparison between multiple objective weighting methods.
- Use of the objective weighting method MEREC [10] to fix the election criteria weights.
- GFEA robustness stems from the integration of several features like reserving a backup leader, improved tie-breaking mechanism that prioritizes criteria, leader recovery, and tolerance to both initiator and leader failure.
- The three correctness properties are proven to be satisfied by the proposed election algorithm, and for the evaluation, GFEA was tested on the collaborative e-maintenance process.
- Comparison of the proposed algorithm with other recent algorithms in terms of functionalities, performance, and results.

This research paper is organized in the following manner. The second section compares multiple distributed election algorithms. The following section introduces the election criteria obtained using the Delphi method, and presents the objective weighting method MEREC used to fix the election criteria weights. The fourth section gives information about the system model, including the topology, assumptions, and notations. Next, the proposed algorithm, GFEA, is detailed in section five. The algorithm's complexity is analyzed in Section six. The seventh section explains the case study of collaborative e-maintenance on which we tested and illustrated the execution of GFEA. Additionally, this section compares different objective weighting methods and discusses the obtained election results. Furthermore, GFEA is compared to other related works based on performance, functionalities, and results. Finally, the paper is concluded, and future directions are explored.

2. Literature Review

This section reviews multiple leader election algorithms designed for distributed systems. One of the newest works is a paper written by Sperling *et al.* [11], which proposed an election algorithm designed for asynchronous distributed systems. This algorithm presents a new voting procedure called shallow ranked voting, which allows the processes to vote for two processes. This algorithm guarantees the privacy of voters. The votes are encrypted using the CKKS (Cheon, Kim, Kim, and Song) method, which is a homomorphic encryption method. Meaning that we can make approximate calculations on the encrypted data without having to decrypt it. This hides the identity of the voters as well as that of the top candidates who are most likely to win the election. If the primary choice doesn't win the election, the secondary choice gets the vote. This is also used for tie-breaking when two or more processes have the majority of votes. To break a tie, the process with the smallest number of votes is eliminated, and its voters' second choices take its votes. Since the UID is not used to break the tie, this ensures the privacy of the top candidates. Nevertheless, this work didn't consider multiple election criteria, a backup, or the recovery of the leader.

Jiang *et al.* [12] proposed a leader election approach based on node weight in the case of split-brain, which is a special case of partitioning when a network is divided into two partitions only. The election starts when the leader doesn't receive a heartbeat signal from the other servers or finds a node with a higher weight. The weight indicates the service level of a node. The leader is the one who gets the majority of votes. The nodes with the minimum weight will ensure the high availability of the system. This approach has less unavailable time than detection node-based and region leader-based approaches. The arbitration program also uses only 2% of the CPU's full capacity. On the other hand, the authors didn't write a formal algorithm, nor did they analyze the time and message complexity of their approach.

Luo *et al.* [13] proposed an algorithm for the election of the block generator in the consensus mechanism of DPoS (Delegated Proof of Stake). They modified the Chang & Roberts ring algorithm by adding Stake Value. During the election, this value will be multiplied by a random value. But if the candidate who sent an election message has already been a leader before. Then, his Stake Value will be multiplied by zero to ensure equality and avoid monopoly. Its message complexity is 2n. Nonetheless, this algorithm doesn't have a tie-break mechanism, nor does it consider a backup leader.

Haddar [14] proposed a scalable and energy-aware k-leaders election algorithm designed for IoT wireless sensor networks. Election starts from the initiators who broadcast an election message that helps to create a tree whose root is the initiator. The initiator receives the possible leaders of its neighbors and sends a Winner message to the k-highest weight nodes and a Looser message to the remaining nodes. The authors compared their algorithm to the other two top K-leader algorithms (WiLE and Top-K). Their algorithm gave better results in the number of messages and bytes transmitted in GRID and fully connected graph topologies with various network sizes. The residual energy was almost the same in the three algorithms. But the authors say their algorithm also consumes less power due to the reduced number of exchanged messages. On the other hand, they did not specify how the weights were calculated for the election criteria. Instead, they used random weights in the experiments.

Cahng and Lo [15] proposed a consensus-based leader election algorithm for wireless Ad Hoc networks, which is based on Bully and Paxos algorithms. It has a fault

detection mechanism through finder nodes for detecting if the leader has left or crashed. The criteria used for election are residual battery power & node degree. The identifier is called "Vote" and is calculated based on these two criteria. The consensus consists of accepting only higher priority proposals and denying others. The proposed algorithm's message complexity is O(n). As future work, they proposed to ensure message integrity. Despite considering more than one election criterion, weights were not assigned.

Raychoudhury *et al.* [16] proposed an algorithm that elects the K-highest weighted nodes as leaders in each connected component of mobile ad hoc networks. The weight of a node indicates its available resources. There are 3 types of nodes: White nodes, which are the normal nodes, *Green* nodes, which are backups for the *Red* nodes, and *Red* nodes, which are the highest-weight neighbors. Red nodes are considered as local coordinator nodes in the sense that they help collect the nodes' weights and forward them to the highest-weighted red node. The red node with the highest weight in a connected component is going to select the top-k nodes based on their weight and elect them as leaders. This algorithm is fault-tolerant and message-efficient. It is also designed with topological changes in mind that could lead to network partitions. Moreover, it reserves a backup leader in case a red node crashes. However, this algorithm is based only on one election criterion, and doesn't consider the recovery of a failed leader.

DRLEF (Distributed and Reliable Leader Election Framework) was proposed in [17] by Elsakaan and Amroun consists of choosing an authentication server from a set of gateways. These gateways coordinate the network of wireless sensors. There are two types of nodes: Gateways and Sensor Nodes. There will be local elections in each area of the WSN (Wireless Sensor Network). The centrality here was measured by the deviation method. If the deviation exceeds a certain threshold, then the GW will not participate in the election. The gateway with the maximum number of GWs as direct neighbors is called the CGW (Central Gateway). Gateways send candidacy messages to the CGW. The candidate GWs are ranked based on the centrality criterion, and the best one is going to be elected as the leader. The other GWs enter hibernation mode, they are kept as backups in case the leader fails. This eliminates the need to redo the election process. However, the DRLEF algorithm does not guarantee election in severe mobility circumstances. Additionally, it doesn't consider multiple election criteria or the leader's recovery.

Julian and Marian Jose [18] used the fuzzy analytic hierarchy process to elect a cluster head in ad hoc networks. The leader is elected based on his weight. The node's weight is calculated based on seven criteria, which are: node degree, transmission range, mobility, residual energy, trust value, status of the node, and fairness of the node. Using this approach has several advantages. First, it eliminates inconsistencies in selection criteria. Secondly, the fuzzy variation of AHP removes duplicate weights. Additionally, it has better performance than the standard WCA (Weighted Clustering Algorithm). On top of that, the nodes' mobility is taken into consideration. Finally, we obtain a ranking of the nodes from best to worst. In contrast, the authors didn't consider the failure of the leader, the recovery of the failed nodes, or the addition of new nodes.

Kadjouh *et al.* [19] presented a dominating tree-based leader election algorithm (DoTRo) designed for smart cities IoT networks. It uses the local minima finding algorithm (MinFind) to discover the local minimum values within the network. Afterwards, each local minimum is going to be the root that initiates a spanning tree. When two spanning trees come in contact, the tree with the smaller value continues the flooding process while the other one stops. Next, the local minima will wait a maximum duration so their flooding processes can end. After this maximum time, if a local minimum node doesn't receive a message, then it becomes the leader. This algorithm is energy efficient, fault-tolerant, and reduces the number of sent and received messages when compared to MinFind. In contrast, this work didn't deal with security issues. Besides, no backup leader was considered, and the election is solely based on one value.

Favier *et al.* [20] introduced a novel centrality-based eventual leader election algorithm that works in dynamic networks. The leader in this algorithm has to be in the center of the network. Each node knows its neighbors and the neighbors of its neighbors. A leader is elected in each component. When a node detects a change in its neighborhood, it updates its knowledge and emits its new view of the network. Nonetheless, there are some drawbacks to this algorithm. Firstly, if the nodes do not have the same knowledge, then they can choose different

leaders. Secondly, this algorithm only considers the criterion of distance between the nodes. In addition, the size of the messages can be important since each node uses map structures, and the message must respect the MTU (Maximum Transmission Unit) of the network packet. This requires compression algorithms to reduce the size of the exchanged messages. It is possible to use collaborative calculations to calculate the centralities and thus save time. Moreover, this work didn't take into account multiple election criteria and leader recovery.

Biswas *et al.* [3] proposed a new resource-based leader election algorithm, which selects the leader based on resource strength. Resource strength is calculated based on three criteria: CPU, memory, and remaining battery for mobile nodes. Each node has a queue with all the nodes present on the system. After that, the queue is sorted in descending order to place the node with the highest resource strength at the beginning of the queue and thus choosing it as the leader. This algorithm also takes into consideration the addition and removal of nodes from the system using update messages. However, the authors did not consider the security aspects of nodes joining the system. Moreover, the frequent addition and removal of nodes slow down the system. Plus, no formal weighting method was specified to determine the importance of each election criterion.

Another work by Biswas *et al.* [21] presented a novel failure rate and load-based leader election algorithm (FRLLE) for bidirectional ring topology in synchronous distributed systems. This algorithm selects the node with the minimum leader coefficient to be the leader. The leader coefficient is computed based on several criteria, which are: average CPU usage, memory usage, bandwidth usage, and failure rate. The elected leader is the node with the minimum failure rate and minimum load, to ensure a stable leader with high performance. The proposed algorithm is faster and exchanges fewer messages than other classical ring-based algorithms. However, the authors didn't specify a formal method to fix the leader coefficient criteria weights, and didn't assign a backup for the leader. Additionally, the authors didn't consider the failure of the initiator nodes. Moreover, the authors considered the best-case complexity when the failed leader recovers, instead of considering the best case of a new election.

Rodrigues *et al.* [22] proposed a new hierarchical adaptive leader election algorithm for static distributed systems that support the recovery of failed nodes. This algorithm is designed for the vCube logical topology. The algorithm selects the process with the smallest UID among the processes that are the most stable. The stability of every process is measured by its incarnation, which is a variable that counts the number of recoveries that a process has had. Initially, all processes elect process 0 as the leader. Afterwards, if the leader recovers after a failure, its incarnation is updated and the second most stable process replaces the leader. A penalty is applied to nodes that fail and recover to avoid being stuck with the same unstable leader. Even though this algorithm brings several contributions, it does not consider multiple criteria.

A recent work by Biswas et al. [23] proposed a leader election algorithm that takes into consideration multiple quality attributes of the leader. This algorithm is made for partially synchronous networks with arbitrary topology, so it doesn't depend on a specific topology. The quality attributes of the leader are determined by human experts. The attributes are fixed by uniting the sets of attributes proposed by each expert. Consequently, these attributes change according to the system requirements. Afterward, the attribute weights are calculated based on the preference of experts using pair-wise comparison matrices that show each expert's preference regarding each pair of quality attributes. The leader is elected based on a score called the quality factor, which is obtained using a modified version of the MCDA method TOPSIS. The algorithm ensures that the system recovers to its correct state after failures and partitioning, thus, it is fault-tolerant. In this algorithm, a node sends election election-initiating message to adjacent nodes, which are considered child nodes. Child nodes send back the maximum quality factor of their children to their parent nodes. A node that receives a message twice becomes a co-parent node. The algorithm required less election time and less communication overhead when compared to the PALE algorithm. It was also compared to other algorithms in terms of functionality and complexity. The multi-attribute algorithm is tolerant to partitioning. On the opposite side, this paper didn't consider assigning a backup leader, meaning, when the leader fails, the election algorithm has to be executed again. In addition, the weighting method used is subjective.

Other classical and popular algorithms like LeLann, LCR, HS, and Bully algorithms [24], [25], [26], [27] rely solely on the UID to determine the leader and don't reserve a backup leader.

Table 1 compares between the previously discussed election algorithms based on functionality and complexity.

Laredj A. *et al.* tackled the problematic of electing a coordinator within a collaborative e-maintenance process [28]. They used the ELECTRE I MCDA method to elect one of the experts as the coordinator. This method resulted in a partial ranking of the experts. Meaning, some experts were incomparable. The experts were evaluated based only on five criteria: experience as an expert, network speed, distance to breakdown site, coordinator experience, and response time. In addition, the weights were fixed without using a formal weighting method. Moreover, using an MCDA method that doesn't provide a complete ranking of the experts resulted in the absence of a backup leader who would handle the process in case the connection between the coordinator and technicians is lost. This requires running another iteration of the election process just to replace the former leader. Furthermore, their work didn't contain visual charts that should help in analyzing and comparing the different experts. Finally, this paper didn't evaluate the performance and quality of using the ELECTRE I method on their example.

Mohammedi Taieb and Laredj [9] proposed a multi-criteria approach for electing the GDSS facilitator. The authors used the Analytic Hierarchy Process to fix the election criteria weights. Then, applied MAUT, SAW, TOPSIS, and PROMETHEE II on a collaborative e-maintenance case study to find the most suitable method among the four for electing a facilitator. MAUT and PROMETHEE II gave similar results. Their comparison showed that PROMETHEE II was easier to use than MAUT and had a better performance.

Table 1. Comparison between existing election algorithms.

Leader Failure Detection	Yes	No	No	Yes	Heartbeat	No	No	Yes	Yes	Yes	Not specified	Yes	Yes, Finder nodes	No	Yes (Heartbeat)	Not specified	Yes	Heartbeat
Leader Recovery	No	No	No	No	No	No	No	No	Yes	Yes	No	No	No	No	No	No	Yes	Yes
Multi Criteria	No	No	No	No	No	Yes	No	No	Yes	Yes	No	No	Yes	No	No	No	UID and Incarnation	Yes
Optimized for Humans	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	No	Nos
Weighting Method	No	No	No	No	No	Yes, Fuzzy Pair-wise comparison	No	No	No	No	No	No	No	No	No	No	No	Yes
Homogeneous	Not Specified	Not Specified	Not Specified	Not specified	Not Specified	Not Specified	Not Specified	Not Specified	Not Specified	Yes	Not specified	Not specified	Not specified	Not specified	Not specified	Not specified	Not Specified	Heterogenous
Partitioning Tolerability	No	No	No	No	two partitions	No	No	Yes	No	No	No	No	No	Yes	Yes	No	No	Yes
Backup Leader	No	No	No	No	No	No	No	No	No	No	No	Yes, Vice gateway	No	No	Yes	Yes	Next most stable	No

Table 1 continued

Table 1 (continued). Comparison between existing election algorithms.

Algorithm\criteria	Time complexity	Message Complexity	Topology	Synchronization	Fault tolerance	Structure	Mobility Support	Election Criteria	Year	Privacy	Tie-Break
LeLann [9]	O(n)	$O(n^2)$	Unidirectional Ring	Asynchronous	Yes	Static	No	UID	1977	No	No possible tie
LCR[10]	O(n)	$O(n^2)$	Unidirectional Ring	Asynchronous	Yes	Static	No	Max or Min UID	1979	No	No possible tie
HS [21]	O(n)	O(nlog(n))	Bidirectional Ring	Asynchronous	No	Static	No	Largest UID	1980	No	No
Bully [22]	O(1)	O(n^2)	Complete Mesh	Asynchronous	Yes	Static	No	Highest priority	1982	No	No possible tie
Weight Based[19]	Not mentioned	Not mentioned	Mesh	Not specified	Only leader	Static	No	Majority weighted votes	2020	No	No
Fuzzy AHP [16]	Not mentioned	Not mentioned	Mesh	Not specified	No	Static	Yes	7 criteria	2021	No	No
DPoS [20]	O(n)	Not mentioned	Unidirectional Ring	Not specified	Yes	static	No	Stake Value	2018	No	No
Centrality-based [18]	Not mentioned	Not mentioned	Arbitrary	Partially Synchronous	Yes	Dynamic	Yes	Closeness Centrality	2021	No	Yes, Highest UID
Resource-based [3]	Not mentioned	O(n)	Unidirectional Ring	Not specified	Yes	Dynamic	No	Resource strength value = CPU, Memory, and Energy (Remaining	2018	No	No
FRLLE [8]	O(n)	$O(n^2)$	Bidirectional Ring	Synchronous	Yes	Dynamic	No	Smallest Leader Coefficient = Failure Rate + Load	2021	No	Yes, Max UID
DotRo [14]	Not mentioned	kn(m+1)	Arbitrary	Not specified	Yes	Dynamic	No	Minimum Value	2020	No	No
DRLEF [15]	Not mentioned	Not mentioned	WSN	Synchronous	Yes	Dynamic	No	Deviation (Centrality)	2022	No	No
Consensus [17]	Not mentioned	O(n)	Partial Mesh/ WADHOC	Asynchronous	Yes	Dynamic	Yes	Remaining Battery & Node Degree	2012	No	No
Privacy-Preserved [11]	Not mentioned	O(diameter(G)* N*p)	Graph (Mesh)	Asynchronous	Yes	static	No	Most Votes	2023	Yes	Yes, secondary
Top-K [12]	Not mentioned	Not mentioned	Arbitrary	Asynchronous	Yes	Dynamic	Yes	Node Weight	2013	No	Yes, Max UID
SEALEA [13]	Not mentioned	Not mentioned	Grid & Complete Graph	Asynchronous	Leader only	Static	No	Node Weight	2022	No	Yes, highest UID
Hierarchical Adaptive	Log2(n)	Nlog2(N) per testing round	vCube	Partially Synchronous	Yes	Static	No	Most stable with smallest UID	2024	No	UID
Multi-attribute	O(n.D)	O(n.l)	Arbitrary	Partially Synchronous	Yes	Static	No	Quality attributes	2025	No	Yes, min ID

3. Methodology and Election Criteria

The GDSS facilitator has to make sure that each participant expresses their worries. Additionally, he/she guarantees that the opinions of the DMs are mapped correctly. Furthermore, the facilitator has to make certain that the methods and procedures are used faultlessly while considering potential human biases. Moreover, he/she informs the participants of the implicit assumptions related to the used methods. Finally, the outcome of the decision model has to be clarified to the DMs by the facilitator [29].

Before the group decision-making process starts, the GDSS facilitator has to create the meeting agenda and select the DMs who will participate in the meeting. Throughout the meeting, the facilitator will be responsible for the generation, organization and evaluation of the alternatives. After the meeting is done, he/she will present the final decision to all the participants [30].

Distributed leader election algorithms are designed to solve the problem of choosing a unique node to be the leader of a connected network [14]. Similarly, the problematic treated in this paper consists of choosing a single DM to be the facilitator of a group of DMs utilizing the GDSS. In both problematics, there are multiple entities (nodes and DMs) and one controlling entity (leader/facilitator). Furthermore, the entities in both fields are geographically distant and connected via a network. Moreover, in both cases, they can communicate with each other via messages. Additionally, the same network protocols can be used in both cases. Figure 1 summarizes the similarities between the two problematics.

Election algorithms and the election of a facilitator have the same goal, which is to agree on a single leader. Both cases require considering certain criteria during the election. However, the criteria are mostly different since one side concerns machines while the other involves humans. Election algorithms have to be fault-tolerant, the same as a facilitator needs a backup DM in case he loses connection with the other group members.

These similarities make the election algorithms seem like a potential solution to the problematic of electing a GDSS facilitator. However, due to the different nature of entities and context, an election algorithm designed for machines has to be modified to support human elections.

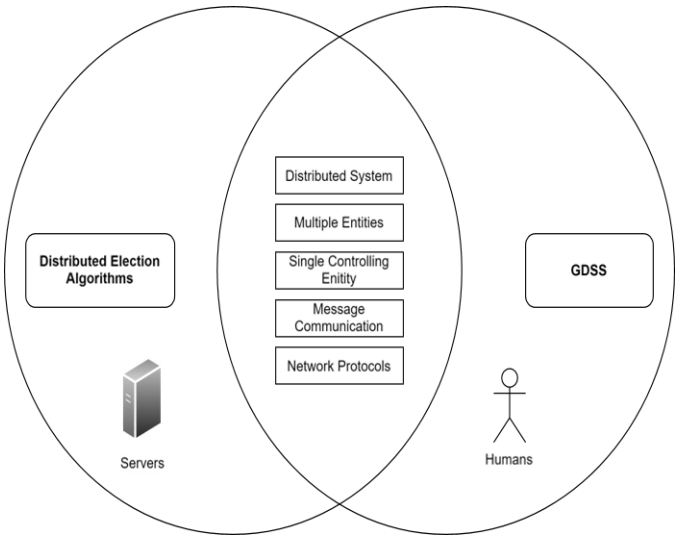

Figure 1. Venn Diagram showing the similarities between distributed election algorithms and GDSS.

The election algorithms that exist in the literature (See literature review in section 2) didn't include all the features required for the GDSS facilitator election in one single algorithm. Consequently, a potential solution is to combine each feature from each algorithm into a new election algorithm designed specifically for the problematic of GDSS facilitator election. A fulfilling algorithm should have a backup leader and should consider multiple criteria that are relatable or applicable to humans. Additionally, it should use a proper weighting method to indicate the importance of each election criterion. Furthermore, it should consider the recovery of failed nodes, because realistically, DMs can lose their connection to the GDSS at any time, and recover their connection later on. Finally, in case a tie happens at the end of the election, a tie-break mechanism is necessary to satisfy the uniqueness property.

This paper introduces a new election algorithm called GFEA, optimized for the election of a GDSS facilitator. In addition, an objective weighting method is used to fix the election criteria weights. Moreover, the failure and recovery of the initiator and the facilitator is handled. Finally, a new tie-breaking mechanism is proposed.

3.1. Election criteria

Gathering facilitator election criteria from DMs requires specifying a real-world problem on which we can use the GDSS. Among the applicable fields is the collaborative e-maintenance process, which was used as a

case study (see section 7.1). The DELPHI method [31] was applied to gather the criteria for electing a GDSS facilitator from industrial maintenance professionals. The election criteria fall into 3 main categories, which are the DM experience, machine security, and network performance. All election criteria, apart from the number of treated breakdowns, are general GDSS facilitator criteria. However, "Treated breakdowns" is a criterion specific to maintenance-related fields such as industrial maintenance. The election criteria are described as follows:

1- **Experience as a DM:** Number of times the DM was involved in a group decision-making process (beneficial).
2- **Treated breakdowns:** This criterion is specific to the collaborative e-maintenance process. It represents the number of repaired breakdowns (beneficial).
3- **Distance:** This criterion measures how far the DM is from the breakdown site in Kilometers (non-beneficial).
4- **Response time:** Elapsed time in minutes between the announcement of a breakdown and the reception of the DM response (non-beneficial).
5- **Coordination experience:** Number of times a DM has participated as a coordinator (beneficial).
6- **Open ports:** Number of open network ports on the DM's machine (non-beneficial).
7- **Vulnerabilities:** Number of vulnerabilities present on the DM's machine, obtained using a vulnerability scanner (non-beneficial).
8- **Severity sum:** Summation of the severities corresponding to the detected vulnerabilities. The severity is represented by the CVSS (Common Vulnerability Scoring System) score of the vulnerability (non-beneficial).
9- **Connection Type:** of the DM's internet connection. There are 4 possible types: mobile, ADSL, Satellite or Fiber. Each one is represented using a numeric score based on its stability, speed, and network delay. Mobile corresponds to 0.4, ADSL: 0.6, Cable: 0.7, Satellite: 0.8, and Fiber is considered the best type of internet connection [32] with a score of 1.
10- **Network latency:** Average network response time between the DM's machine and the breakdown site, measured in milliseconds (non-beneficial).
11- **Download speed:** Download speed of the DM's internet connection in Mbps (beneficial).
12- **Upload speed:** Upload speed of the DM's internet connection in Mbps (beneficial).

3.1.1. Weighting method

There are three types of weighting methods in the literature: objective, subjective, and combined. When using objective weighting methods, the DMs' subjective preferences are not considered [33]. In this work, the authors opted for objective methods to keep the election algorithm formal and unbiased. Among the popular objective methods are: the entropy, mean weight, CRITIC, standard deviation, and statistical variance methods [34]. There are other newer methods like CILOS, IDOCRIW, and MEREC. In MEREC (Method based on the Removal Effects of Criteria), the effect of removing a criterion on the evaluation of alternatives is used to fix the criteria weights [10]. The MEREC weighting method was used in this paper to fix the election criteria weights.

Among the reasons MEREC was chosen instead of other objective methods is that it is easy to understand and use, since it simply involves applying a set of formulas to the values that are already present in the performance matrix. In addition, it has a strong mathematical foundation [35]. However, this doesn't mean that other objective weighting methods aren't valid. It is possible to use other methods with the proposed algorithm.

This method takes the performance matrix and the criteria types as input. The performance matrix should only contain positive, non-zero values.

The values corresponding to non-beneficial criteria are normalized according to formula (1). While in the case of beneficial criteria, the values are normalized by applying formula (2) [10].

$$x'_{ij} = \frac{x_{ij}}{\max_i(x_{ij})} \quad (1)$$

$$x'_{ij} = \frac{\min_i(x_{ij})}{x_{ij}} \quad (2)$$

Next, the general performance GP_i of each alternative i is calculated based on formula (3) [10].

$$GP_i = \ln(1 + (\frac{1}{m}\sum_j |\ln(x'_{ij})|)) \quad (3)$$

The following step consists of constructing m sets that contain the performance of alternatives when removing each criterion j. This performance is calculated using formula (4) [10]. Where k represents all criteria except criterion j.

$$GP'_{ij} = \ln(1 + (\frac{1}{m}\sum_{k,k\neq j} |\ln(x'_{ik})|)) \quad (4)$$

In the succeeding step, the removal effect of each criterion is calculated using formula (5) [10].

$$RE_j = \sum_{i=1}^{n} |GP'_{ij} - GP_i| \quad (5)$$

Finally, the criteria weights are obtained from the removal effects using formula (6) [10].

$$w_j = \frac{RE_j}{\sum_{k=1}^{m} RE_k} \quad (6)$$

4. System Model and Problem Definition

4.1. Problem definition

Given a set D of DMs connected using a unidirectional ring network with |D| = n. One of the DMs has to take the role of the GDSS facilitator [4], while another one is reserved as a backup. The latter replaces the failed facilitator when he loses his connection with the network. The election algorithm has to consider multiple election criteria, and a formal weighting method is needed to distinguish the importance of each criterion. Additionally, the system has to be fault-tolerant and must be capable of handling the potential recovery of failed nodes. Finally, the election algorithm must satisfy the 3 correctness conditions (uniqueness, termination, and agreement) [21].

4.2. System model

The network is a synchronous, static, unidirectional ring composed of n nodes. Each node represents a DM. There has to be at least 2 decision-makers in the network (n>=2) [4]. Message passing is used for communication. The message delay between two nodes is based on the distance between the two DMs. If a DM loses his connection with the network, then its node is considered a failed node.

4.3. Assumptions

To simplify the implementation and analysis of the proposed algorithm, a set of assumptions was considered. First, each node has a unique identifier (0 < UID <= n) [21], which also indicates the order of joining the decision-making session (First DM to join the session has the smallest UID). Moreover, nodes are homogeneous and not mobile. Furthermore, the maximum number of nodes n is fixed before starting the election. Additionally, each node knows the UID of the previous and next adjacent nodes [16]. Another assumption is that each node receives a heartbeat from the previous node and sends a periodic heartbeat message to the next adjacent node to detect if the previous node fails [16]. In addition, the communication direction is clockwise. On top of that, the DM's performance is still used in calculating the election criteria weights even if the DM loses his connection, because this gives more input data to the objective weighting method. Furthermore, there are no hops between each pair of adjacent nodes. Moreover, no new DMs are added to the network other than the preselected n DMs. Finally, the ring topology is only used for the election and failure tolerance, it is not used for the actual group meeting communication.

4.4. Notations

There are 7 types of messages in this algorithm. The description of each type is presented below:

- **Message(uid, value[j]):** General message object containing the UID of the message creator and his value of the jth criterion.
- **InitiationMsg(uid, value[1]):** Initiation message created by the initiator, which includes the UID of the initiator node. In addition to its value of the most important criterion (1st criterion).
- **LeaderMsg(leader_uid, backup_uid):** Message announcing the new elected leader and the backup leader to all other nodes.
- **FailureMsg(uid):** Message announcing the failure of a node by sending its UID.
- **LeaderFailureMsg(new_leader_uid):** Message announcing the leader failure, and informing other nodes that the backup has become the new leader.
- **RecoveryMsg(uid):** Message announcing the recovery of a previously failed DM by sending his UID.
- **LeaderRecoveryMsg(failed_leader_uid):** Message announcing the recovery of the previously failed leader, thus updating the nodes with his new state, and informing them that the leader has become a backup.
- **TieMessage(ties[k], uid, value$_a$[m]):** Message containing the UIDs of the tied DMs holding the maximum score, the UID of the message creator, and their values of the *j*-th criterion.

Variables used within GFEA are detailed below:

- **state[n]:** List containing the combination of role (DM, Initiator, Leader) and state (Failed or

Connected) of each node in the network. Each node has its local *state* list, which gets updated when receiving messages. This allows each node to be aware of the current state of every other node in the same network.

- **value[m]:** List containing the evaluation of a DM in each election criterion.
- **criterion_type[m]:** List containing the type of each criterion, either a beneficial (maximization) criterion or non-beneficial (minimization) criterion.
- **r:** Received message.
- **score[n]:** List containing the algorithm score for each DM.
- **best_dms:** List of DMs with the maximum score.
- **ranking[n]:** List containing the ranking of DMs based on the final score.

Procedures integrated within the proposed algorithm are as follows:

- **send(Message msg):** Procedure for passing a message to the next adjacent node.
- **broadcast(Message msg):** Procedure implying that each node should keep forwarding the contained message to the next node until it reaches its original node.

5. Proposed Election Algorithm

In this work, a new election algorithm called GFEA (GDSS Facilitator Election Algorithm) is proposed. This algorithm is inspired by the FRLLE election algorithm [21] and MCDM (multi-criteria decision-making) methods. This algorithm is optimized for the problematic of electing a GDSS facilitator. It uses the ring topology with unidirectional communication channels. The nodes communicate via message passing. Each DM has a unique identifier that indicates the order in which the DM joined the decision-making session. Furthermore, each node is in one of seven states: Initiator, DM, leader, backup, failed leader, failed initiator or failed DM. Flowchart Figure 2 illustrates the proposed leader election algorithm concisely.

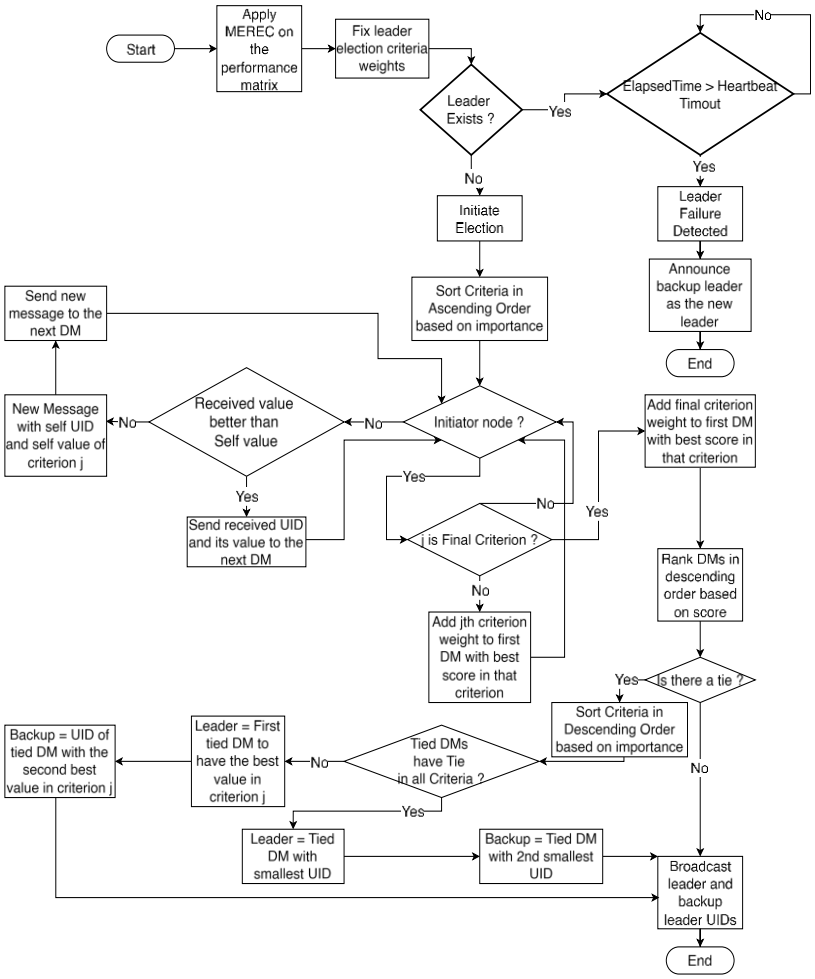

Figure 2. Flowchart Summarizing the Proposed Election Algorithm GFEA.

Initiation Phase: This phase concerns the detection of the failed leader and the composition of the first election message. If a facilitator already exists and the node that is next to the leader doesn't receive a heartbeat message from him within a period called the heartbeat timeout [36], it announces that the leader has failed, and the backup leader becomes the new leader. However, if there is no leader in the network, then the first DM to join the session (UID = 1) becomes the initiator of the facilitator election. The first-ever election only starts when all DMs have joined the session. The initiator starts by sorting the election criteria based on their importance in ascending order (From least important to most important). Next, he creates and sends the election initiation message containing his UID and its value of the 1st most important criterion.

Scoring Phase: In this phase (Algorithm 1), when a node receives an election message, it checks its value in criterion j. If the received value is better than its own (greater in the case of maximization criteria and smaller in minimization criteria), and then forwards the message to the next node. But if the received value is worse than its value, it creates a new message containing its UID and its value of criterion j. Then it sends the message to the next node. Once the message reaches the initiator node, it sends a new message for the next most important criterion, j+1. The same process is repeated for all election criteria. If more than one DM has the best value in a criterion, then only the DM with the smallest UID gets the criterion weight added to his score. At the end of the final round, if multiple DMs have the maximum score, then a tie-break mechanism is used to return a single facilitator (see Algorithm 2). Afterward, the initiator node sends the chosen leader and the backup leader UIDs in a broadcast message to inform all other DMs who the current facilitator is and who should replace him if he fails.

Tie Break: In case two or more DMs have the same final score, then the initiator uses Algorithm 2 to break the tie and return one DM as the facilitator. First, the initiator starts by sorting the criteria in descending order based on their weights. Starting from the most important criterion (biggest weight) to the least important criterion (smallest weight), the first DM to have a better value than all other tied DMs in a criterion j is declared the leader. The second-best value in the same criterion j is selected as the backup leader. If multiple DMs have the best value in a criterion j, then the algorithm continues to the next most important criterion and checks again. In the rare case of having a tie in all criteria, then among the DMs having the maximum score, the first one to have joined the session (smallest UID) is elected as the facilitator, and the one who joined after him is the backup leader (second smallest UID). The tie-break mechanism is detailed in Algorithm 2.

Handling Leader Announcement Message: When a DM node receives the leader announcement message, it updates its state list with the new leader and backup leader UIDs and passes the received message to the next node. On the other hand, if the initiator node receives this type of message, it changes its state to DM. Except if it's already a leader or a backup leader, in that case, it doesn't change its state.

Algorithm 1. Scoring Phase

```
Input: pm, criteria_weights, criterion_type.
Output: leader, backup.
n: number of decision makers;
m: number of criteria;
r: received message;

1   If(this node is the initiator node) Then
2     If(this is the final round) Then
3           score[r.uid] = score[r.uid] + criteria_weights[j];
4           best_dms = UIDs of DMs with max score;
5           If(Only one DM has the max score)Then
6              leader = best_dms;
7                 ranked = sort DMs from highest score to
    smallest score;
8              backup = ranked[2];
9              leader_msg = new LeaderMsg(leader,
                backup);
10             broadcast(leader_msg);
11          Else tie_break(best_dms);
12      Else Add weight of j to the score of the
          received UID;
13          msg = new Message(self.uid, value[j+1]);
14          send(msg);
15  Else If(criterion_type[j]==”max”) Then
16        If(r.value[j] < self.value[j]) Then
17           msg = new Message(self.uid,
             self.value[j]);
18           send(msg);
19        Else  send(r);
20  Else If(criterion_type[j]==”min”) Then
21          If(r.value[j] > self.value[j]) Then
22              msg = new Message(self.uid,
                self.value[j]);
23              send(msg);
24          Else  send(r);
```

Algorithm 2. Tie Break Between Multiple DMs

```
ties = [a, b, …, k]; // k tied nodes
sortDescending(criteria, criteria_weight);
j = 1;
If(this is the initiator node)Then
  While(j <= m) Do
    If(initiator is one of the tied DMs)Then
      tie_break_msg = new TieMessage(ties, self.uid, self.value[j]);
    Else tie_break_msg = New TieMessage(ties, self.uid, Null);
    send(tie_break_msg);
Else If(this is a DM and criterion_type[j]==”max”)
    If (this is a tied DM and r.value[j] < self.value[j] or r.value[j] == Null) Then
      self.backup = r.uid;
      self.leader = self.uid;
      found = True;
      tie_break_msg = new TieMessage(ties, self.uid, self.value[j], found);
      send(tie_break_msg);
    Else send(r);
Else If(this is a DM and criterion_type[j]==”min”)
    If (this is a tied DM and r.value[j] > self.value[j] or r.value[j] == Null) Then
      self.backup = r.uid;
      self.leader = self.uid;
      found = True;
      tie_break_msg = new TieMessage(ties, self.uid, self.value[j], found);
      send(tie_break_msg);
    Else send(r);
Else If(This is a DM)Then
      send(r);
If(Initiator receives TieMessage and found == True)
      leader = r.leader;
      backup = r.backup;
      leader_msg = new LeaderMsg(leader, backup);
      BreakLoop;
Else If(Initiator receives TieMessage and found == False)
      j = j + 1; // Continue to next criterion
Else If(there is a tie in all criteria) Then
      leader, backup = best_dms[1], best_dms[2]; //Choose 2 DMs with smallest UIDs from tied DMs
      broadcast(leader_msg);
```

Fault Tolerance: In case the facilitator gets disconnected from the network, the node next to the failed leader that detected his failure sends a leader failure message containing the UID of the backup as the new leader. The backup leader is the DM with the second-best score. Having a backup eliminates the time and resource cost of running the election another time when the leader fails [14]. If a DM loses his connection with the network, then his state is changed to “Failed Node”, and his previous node gets connected directly to his next node in order to keep the ring topology intact. This network doesn’t support partitioning as it will always try to keep its logical ring topology intact.

Additionally, if the initiator node fails in the first round (j = 1), then the node next to the failed initiator becomes the new initiator, and the election continues without interruption (see Algorithm 3). On the other hand, if the initiator fails after the first round, then the election has to restart, and the node next to the failed initiator becomes the new initiator. Because the failed initiator had the list containing the scores. Hence, the new initiator creates the election initiation message starting with the most important criterion and sends it to the next node.

Algorithm 3. Initiator Failure

```
If(Initiator node fails”) Then
  If(it’s the first round) Then
    Node next to the failed initiator becomes the new initiator;
    Continue current election;
  Else Stop current election;
    Node next to failed initiator becomes the new initiator;
    i_msg = New InitiationMsg(new_initiator, value[1], new_initiator);
    send(i_msg); \\ New initiator sends the new initiation message.
```

Failure Recovery: When a previously failed node joins the network, it restores its previous status (DM or Leader). As a result, if the backup has already replaced the leader and the previously failed leader gets reconnected to the session, then he becomes a leader again, and the backup becomes a backup again (see Algorithm 4). Next, the recovered leader sends a recovery message to the other nodes. Furthermore, if a DM recovers before the final round (criterion m) is finished, he is still considered a candidate. Because the criteria are sorted in ascending order, he can compensate for the previous rounds (less important criteria) by scoring in the most important criteria (later rounds). However, if he recovers after all the rounds have gone through, then he isn’t considered a

candidate and will be removed from the ranking, as the initiator is already in the leader announcement phase and isn't aware of the DM recovery until the recovery message reaches him.

If the failed initiator recovers during the first round, then it restores its state as the initiator, and the election continues. But if the first round has passed, then it becomes a DM.

When the leader receives his recovery message, he discards it [21]. However, if a DM node receives a leader recovery message, it updates its state list with the new state of the recovered leader and forwards the received message to the next adjacent node.

Algorithm 4. Recovery of failed nodes

```
If(This is the recovering leader) Then
    backup = leader;
    leader = previously failed leader UID;
    rec_msg = new LeaderRecoveryMsg(leader, backup);
            broadcast(rec_msg);
Else If(This is the recovering initiator and current round == failure round) Then
    Recovered initiator restores his initiator state;
    Current initiator becomes a DM;
    continue_election();
Else If(This is the recovering initiator and current round ≠ failure round) Then
    Recovered initiator becomes a DM;
    New initiator continues current election;
Else        // This is a DM
    rec_msg = new RecoveryMsg(self.uid);
    broadcast(rec_msg);
```

5.1. Election algorithm correctness

For an election algorithm to be correct, it has to satisfy the three following conditions [21].

Uniqueness: The DM with the highest score will be elected as the GDSS facilitator. However, if there are multiple DMs having the same max score, a tie-breaking mechanism is used. Starting from the most important criterion to the least important one, the first candidate to have an advantage in criterion j will be selected as the leader. Which means that there will always be one single leader in the system.

Termination: The algorithm takes $m \times n + n$ time steps when there is no tie. In contrast, in the worst-case scenario, it takes $m \times n + m \times k + n$ time steps when there is a tie, where k is the number of tied DMs. Consequently, the algorithm does terminate in a finite time.

Agreement: At the end of the algorithm or after the tie-breaking mechanism ends, an announcement message containing the elected leader and backup leader UIDs is sent to all DMs. Thus, every DM in the group is aware of the newly elected facilitator.

6. Complexity Analysis

In this section, the proposed algorithm GFEA is analyzed based on the number of time steps required and the total number of exchanged messages in both best and worst cases.

6.1. Time complexity

Time complexity is determined based on the number of time of steps that the election algorithm takes to complete.

Best case: When the leader fails, the node next to it sends a failure message containing the UID of the backup, which goes through n-1 nodes. Thus, the proposed algorithm takes n-1time steps to end. As a result, the best time complexity for the proposed GFEA algorithm is: $\Omega(n)$. Where n is the number of DMs.

Worst case: The worst time complexity is when n-2 initiators fail after initiating all 12 rounds and before receiving the final round message. In addition to a tie in the score and all the tied DMs have the same values in all criteria. An additional 2m time steps are required to break the tie (two tied DMs). So, the total number of time steps is: $(n-2)\times((n+2)/2) + 2m + 2 = m(n^2/2 + n) + 4$. Thus, the worst time complexity is: $O(m \times n^2)$.

6.2. Message complexity

Message complexity is obtained based on the number of sent and received messages between all the nodes during the election.

Best case: The best-case scenario is when the leader fails and the failure message informs all n-1 nodes that the backup is replacing him. Therefore, in the best case, 2n messages are exchanged. Consequently, the message complexity of GFEA in the best case is $\Omega(n)$.

Worst case: The worst theoretical case is when n-2 initiators fail after initiating all 12 rounds and before receiving the final round message. In addition to a tie in the score and all the tied DMs have the same values in all criteria. As a result, the proposed algorithm exchanges $2((n-2)\times((n+2)/2) + 2m + 2) = m(n^2 + 2n) + 8$ messages. Therefore, the message complexity of GFEA in the worst case is $O(m\times n^2)$. An advantage of this algorithm is that there is only one initiator at a time. Thus, it avoids multiple nodes initiating the election at the same time.

7. Empirical Assessment

7.1. Case study

The proposed algorithm was tested on the case of collaborative e-maintenance process in an industrial setting. The collaborative e-maintenance system is a distributed system that connects multiple experts distant from each other to enable the proper coordination of their tasks and output a solution in the form of a list containing the repairing actions [37]. Similarly, the GDSS also connects several DMs distributed geographically to reach a consensus. By analogy, GDSS can be used in the collaborative e-maintenance process to conduct the decision-making process, resulting in choosing the corrective actions to repair a broken industrial machine. Furthermore, the collaborative e-maintenance requires choosing one of the experts as the coordinator [37], who has similar tasks to the GDSS facilitator. Since they are both responsible for preparing and coordinating the group meeting.

Besides discussing role assignments with the other experts, the maintenance coordinator prepares and coordinates the work. Additionally, he ensures that the delays are not ignored [37]. Moreover, when there is an additional breakdown, he checks whether or not there is an idle expert or an idle group. If an expert is idle, the coordinator then assigns him to the group treating the new breakdown. Furthermore, when a group of experts needs resources, the coordinator checks if the resources are used or if there is a previous higher-priority request. If the resource is idle, then he grants the permission to the group, otherwise, he puts them in a queue. In addition, the coordinator has to give each expert a sequence number. On top of that, he can send an invitation to an idle expert to join a group of experts. Finally, he verifies the availability of the experts during the creation of a new group [38].

When a breakdown happens, the technician on site notifies the expertise center. The center then selects multiple experts who are usually geographically distant from the site. The number of selected experts depends on the number of fields and the severity of the breakdown.

During our case study, an industrial machine stopped functioning correctly. Consequently, the expertise center invited six experts to diagnose and provide a list of repair actions to the technician on site. However, a coordinator is required to proceed with the maintenance process. Here, the coordinator is also going to be the facilitator of the GDSS. The six experts were evaluated based on the 12 election criteria, which resulted in constructing the following performance matrix Table 1.

7.2. Comparison of objective weighting methods

Table 2 shows the execution time in milliseconds (ms) of five objective weighting methods on our performance matrix (Table 3) using their Python implementations. The authors used the “objective weighting” Python package as an implementation of the methods. Figure 3 illustrates the obtained weights from every method using a bar chart.

Table 2. Execution time of objective weighting methods.

Method	MEREC	CILOS	IDOCRIW	CRITIC	ANGLE	Entropy
Execution Time (ms)	0.97	0.96	0.97	7.01	0.96	1

MEREC was applied to the performance matrix. As a result, the weights corresponding to each election criterion were fixed. The importance of each criterion is presented in Table 4 and is illustrated in Figure 3. From Table 4 and Figure 3 it is observed that applying MEREC resulted in assigning the coordination experience criterion greater importance than all other election criteria.

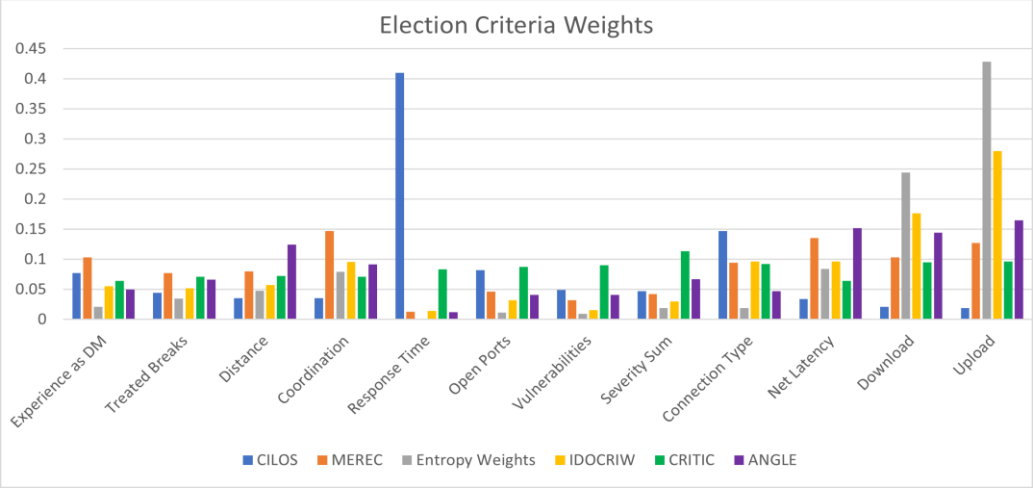

Figure 3. Bar chart showing election criteria weights using different objective weighting methods.

Table 3. Case study performance matrix

	Experience as DM	Treated Breaks	Distance	Coordination	Response Time	Open Ports	Vulnerabilities	Severity Sum	Connection Type	Net Latency	Download	Upload
DM 1	12	8	500	2	17	20	16	32	1	13	100	10
DM 2	37	30	3938	19	20	15	14	10	0.8	366	25	2.5
DM 3	44	23	4401	13	19	18	20	22	0.6	486	50	5
DM 4	29	11	3477	5	18	9	22	31	0.3	198	20	2
DM 5	38	10	5029	3	17	11	10	19	1	103	500	200
DM 6	24	18	1846	9	21	13	18	28	0.8	232	20	2

Table 4. Election Criteria Weights Using MEREC

Criteria	Experience as DM	Treated Breaks	Distance	Coordination	Response Time	Open Ports	Vulners	Severity Sum	Connection Type	Net Latency	Download	Upload
MEREC Weights	0.103	0.077	0.08	0.147	0.013	0.046	0.032	0.042	0.094	0.135	0.103	0.127

In this case study, it was the first time that the expert group was formed. Meaning that the coordinator (leader) role hasn't been assigned to any expert yet. Consequently, the DM who was the first one to join the group session is the initiator of the leader election. He is also given the smallest UID (DM 1). Here, each node represents the machine of a decision-maker. DM1 sorts the criteria in descending order based on their weights. Next, he creates an initiation message i_msg(1, 2) and sends it to the adjacent node next to him, following the clockwise direction, which is DM 4. The original initiation message contains the UID of the initiator node 1 and his performance on the most important criterion, which is the coordination experience criterion. Figure 4 illustrates how the election algorithm is initiated.

When DM 4 receives the initiation message from DM 1, he checks whether the received value of the first criterion is greater than or equal to his value. DM 4 value is greater than the received one. So, he creates a new message by setting the best value holder parameter to his own UID (4). It updates the best value for that criterion to 5. Then, he sends the updated message $msg_1(4, 5)$ to the next node (DM 6). Eventually, when the message reaches DM 2, who has the best value in that criterion, he creates

a new message in which he changes the best value of the 1st criterion to his value 19 and changes the best value holder parameter to 2. After that, he sends the message $msg_1(2, 19)$ to the next node (DM 5). This can be observed from Figure 4.

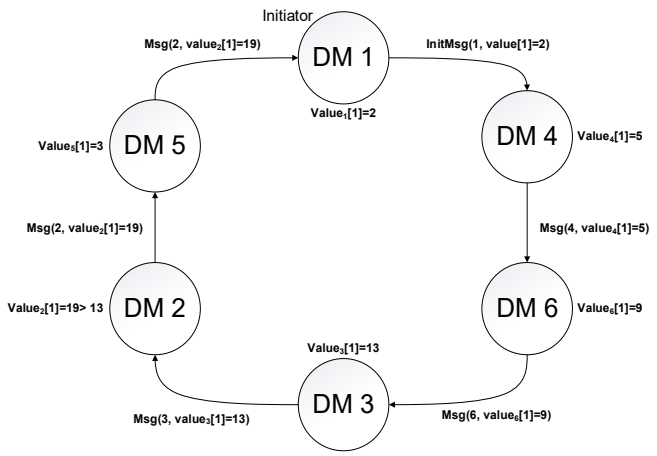

Figure 4. Election initiation and first election round.

When the initiator receives a message containing the index of the same criterion it first sent, msg_j(uid, value[j]). It adds the criterion weight to the DM's score whose UID is in the received message. This process is repeated for all m election criteria. Handling the second round of the election, which concerns the 2nd most important criterion (network latency), is illustrated in Figure 5.

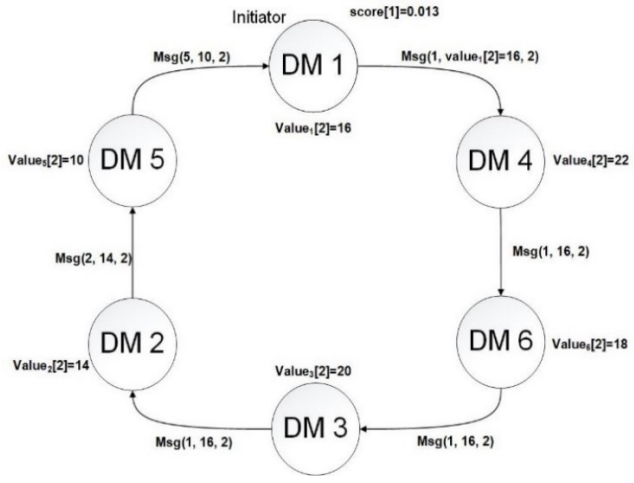

Figure 5. Second round of the election.

When the message of the final criterion, $msg_{12}(1, 17)$, which is the least important criterion, reaches the initiator, it adds the criterion weight to the DM with the best value in that criterion. In addition, it chooses the DMs with the highest and second-highest scores and sends their UIDs in a leader announcement message, leader_msg(1, 2). These two UIDs represent the UIDs of the leader and backup leader, which are 1 and 2, respectively. This phase is illustrated in Figure 6.

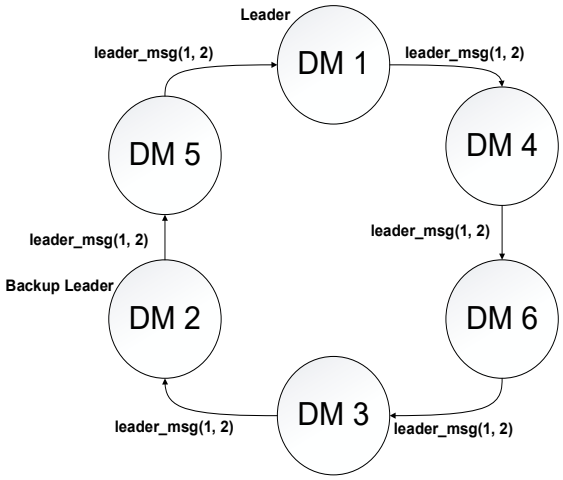

Figure 6. Leader & backup leader announcement.

The final GFEA score of each DM is shown in Table 5. The elected facilitator is DM 1, who was initially the initiator, and the backup leader is DM 2, who got the second highest score. The initiator ranked the DMs, including itself, based on their score. This ranking is presented in Table 6.

Table 5. Algorithm score for each DM.

DM UID	1	2	3	4	5	6
Score	0.322	0.266	0.103	0.046	0.262	0

Table 6. Ranking of the DMs based on GFEA score.

Rank	1	2	3	4	5	6
DM UID	DM 1	DM 2	DM 5	DM 3	DM 4	DM 6

7.3. Leader failure scenario

This subsection illustrates an example scenario in which the leader DM 1 loses his connection to the network. In this case, the next adjacent node, DM 4, doesn't receive a heartbeat message from the leader for a period exceeding the threshold. Consequently, DM 4 considers that the leader has failed. Next, DM 4 sends a leader failure message leader_failure_msg(2) containing the UID of the new leader DM 2 (backup) who replaces the facilitator automatically without executing the election algorithm again (see Figure 7).

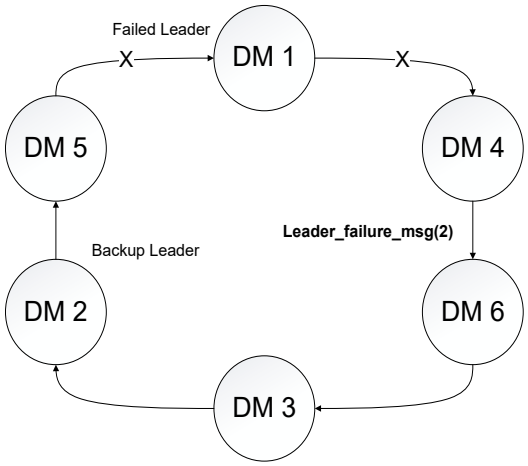

Figure 7. The leader disconnects from the network.

When DM4 receives the leader failure message, it discards it. Finally, each node keeps sending a periodic heartbeat to its next node (see Figure 8).

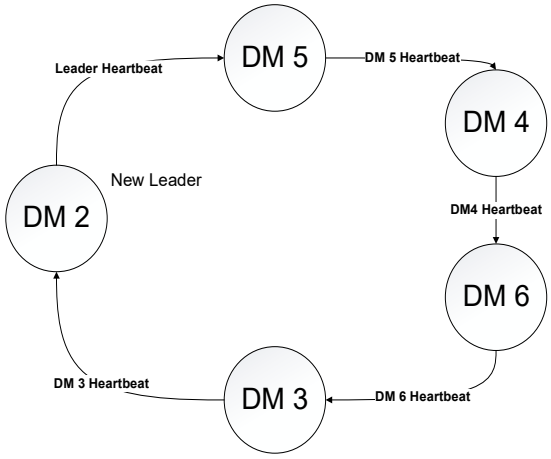

Figure 8. Backup replaces the failed leader.

7.4. Results & discussions

Figure 9 and Figure 10 show the normalized values of the elected leader and backup leader. These values were normalized using the best values available on the performance matrix.

From Figure 9 and Table 1, it is seen that the elected leader (DM 1) is the closest DM to the breakdown site. This can come in handy if an expert physical presence is required on-site or if he loses his connection to the GDSS. Plus, it reduces the cost of time and travel fees for the expert to arrive at the site. Similar to DM 5, he also has the best response time. Additionally, DM 1 has the least network lag (13 ms), which allows him to work with other DMs almost in real time. This is mainly because of two factors: he has the best type of internet connection (fiber optic), and he has the shortest distance to the breakdown site [39]. However, DM 1 doesn't perform well in the security category compared to other DMs. This means that his machine makes the GDSS vulnerable to confidential information leaks and malicious modifications.

Based on Figure 10 and Table 1, the backup leader (DM 2) has the most coordination experience, ensuring a well-planned meeting, better handling of conflicts, and highlighting each DM's opinion using the right questions. Furthermore, DM 2 has participated the most in treating industrial breakdowns and has been an industrial expert longer than DM 1, giving him an edge when it comes to technical questions, fetching required information from the database, and breakdown diagnosis.

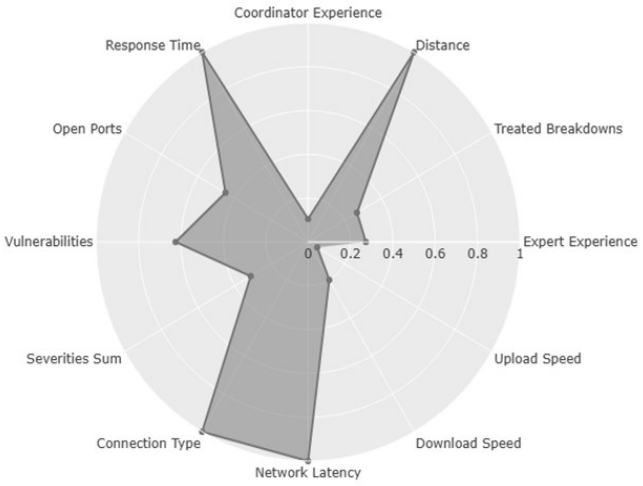

Figure 9 Radar chart showing normalized dataset values of Expert 1.

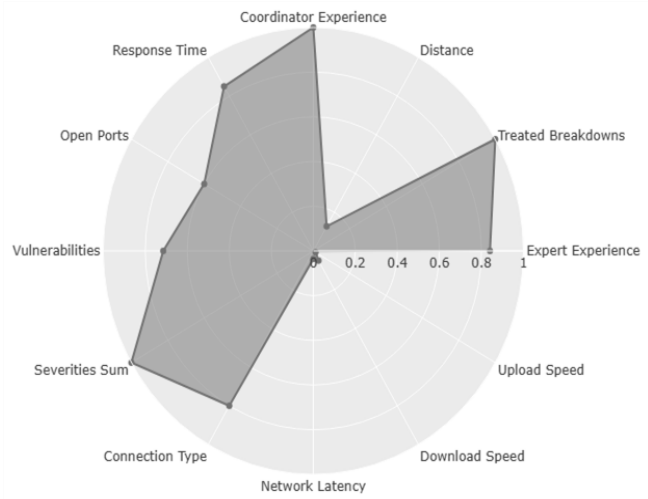

Figure 10. Radar chart of normalized dataset values of Expert 2.

Additionally, in the security category, DM 2 performs better than DM 1. Since his machine has fewer vulnerabilities with less severity and fewer open ports. This means that his machine makes the GDSS less vulnerable to malicious attacks. However, both his download and upload speeds are considerably slower when compared to other experts like DM 5. This can be time-consuming when uploading or downloading files. On top of this, his network delay is slightly high, resulting in a lag during real-time audio and video communications with the technician and with other DMs. Moreover, DM 2 has an average response time when asked to join a decision-making session.

7.5. Comparison with other election algorithms

Table 7 compares GFEA with other classic ring election algorithms using our performance matrix based on execution time, number of exchanged messages, and maximum resident memory. These algorithms were tested using MPI4Py on a local machine composed of a 64-bit Intel 6600HQ processor and 16 GB of RAM.

Table 7. Performance comparison.

Election Algorithm	Execution Time (ms)	Number of Exchanged Messages	Max Resident Memory (MB)
GFEA	131	154	35
Ring	172	24	34.56
LCR	147	34	34.79
Bully	21	96	18.38

Table 8 compares the proposed GFEA algorithm with modern election algorithms that exist in the literature. The comparison is based on the functionality it satisfies, the time complexity of the worst case in Big O(n) notation, the message complexity in the worst case, the UID of the elected leader, and the UID of the backup.

From Table 7 we can see that GFEA is faster than Ring and LCR, however, it is slower than Bully. GFEA also consumes more memory and exchanges more messages than other algorithms. This is mainly because GFEA goes through 12 rounds as it considers multiple election criteria rather than just basing the election on the UID, like the Ring, LCR, and Bully. The compared algorithms apply to human elections. However, they are not optimized for humans. Because the criteria considered in these algorithms are not human-related. This would result in choosing a facilitator solely based on their machine's performance while ignoring experience and security criteria, which is not suitable for the facilitator role. The privacy-preserving election algorithm [11] uses votes to determine a leader. This is a common approach used in human elections. However, it's still very limited and subjective.

The GFEA algorithm showcases the strengths and weaknesses of each DM. Because contrary to algorithms like FRLLE and Resource-based [3], [21] it doesn't combine all criteria into one single value before the election, but rather builds the score progressively during the election. This is because the score can't be calculated solely based on the individual list, instead, the best value across all nodes is needed to determine the best node in each criterion, and then its weight is added to the score. This represents an advantage to GFEA, as it uses a global view instead of a local one.

Table 8. Comparison of the proposed election algorithm with existing algorithms.

Algorithm	Satisfied Functionality	Worst Time Complexity	Worst Message Complexity	Elected Leader	Backup Leader
GFEA	All	$O(mn^2)$	$O(mn^2)$	DM1	DM2
Ring [24]	None	$O(n^2)$	O(n)	DM6	No
LCR [25]	None	$O(n^2)$	O(n)	DM6	No
Bully [27]	None	O(1)	$O(n^2)$	DM6	No
FRLLE [21]	Recovery	O(n)	$O(n^2)$	DM1	No
Top-K [16]	Backup	Not Specified	Not Specified	DM6	DM3
SEALEA [14]	Backup	Not Specified	Not Specified	DM6	DM3
DRLEF [17]	Backup	Not Specified	Not Specified	DM3	DM5
Privacy-Preserved [11]	Tie-break	Not specified	O(d(G).n.p)	DM4	No
Multi-attribute [23]	Multi-criteria	O(n.D)	O(n.l)	DM2	No

Additionally, no fictitious reference values are used for defining what's the best value in each criterion. As a replacement, the actual best values present within the performance matrix are used as a reference. Another plus that puts GFEA in a more favorable position is that, instead of using random weights or weights fixed by human judgment, like in FRLLE and Resource-based [3], [21], it uses MEREC [10] which is a formal objective weighting method. The weights affect the final score of nodes; thus, they can alter the elected leader and backup leader. Furthermore, in contrast to the existing election algorithms, the proposed algorithm considers the failure and recovery of the initiator node. This aspect is neglected in most of the related works. Moreover, the tie-break mechanism integrated within GFEA considers the election criteria instead of just relying on the UID. Plus, it gives priority to the most important criteria. This ensures that the proposed algorithm always elects the most suitable node for the leader position by satisfying the election criteria.

If LeLann, LCR, Bully, or HS algorithms [24], [25], [26], [27] were applied to our case study, then DM 6 would have been elected as the leader. Despite DM 6 not having the best value in any criterion and having the worst score (zero score), he would still be elected in UID-based algorithms, because he has the biggest UID. Additionally, if the smallest UID was considered as the election criterion, then DM 1, who is the most suitable expert for the role, would win the election. However, this approach is not reliable, as the UID in our case indicates the joining order. Meaning that if DM 6 was the first to join the session, then he would be elected as the leader despite him being the worst fit for the leader role. Consequently, the UID alone cannot be the determining factor for the suitability of a DM for the leader role. Additionally, in GFEA, only one node can detect the leader failure, and only one node can initiate the election at a time. This is a considerable advantage when knowing that in classical ring election algorithms, multiple nodes can detect the leader's failure and initiate multiple elections at the same time [25].

8. Conclusion & Future Directions

This work introduces a new distributed leader election algorithm designed specifically for electing a human GDSS facilitator. The system considered is a fault-tolerant unidirectional ring synchronous system, in which each node represents a DM, and they communicate via message passing. The proposed algorithm integrates multiple election criteria falling into security, experience, and network performance categories. These criteria are weighted using the MEREC method. Furthermore, GFEA reserves a backup leader to replace the facilitator in case the connection fails between the facilitator and the GDSS. This saves time and avoids halting the group decision-making session. Additionally, this algorithm doesn't use the UID to break a possible tie. Instead, it uses a new tie-breaking mechanism that searches for the DM who has an advantage in the most important election criteria. Moreover, the failure and recovery of both the initiator and the leader are handled efficiently. No existing election algorithm has integrated all these features. Finally, GFEA is flexible and can be applied to the classical leader election problem in distributed systems by changing the election criteria to machine-related criteria, such as CPU and memory load.

The proposed Algorithm has certain limitations. Its robustness compromises performance and adds network overhead. GFEA is much slower and uses many more messages when compared to other classical ring algorithms, such as Ring and LCR. Mainly because of considering multiple election criteria and taking into account the failure of the initiator. However, the performance can be improved by optimizing the algorithm and reducing its time and message complexity.

This paper identifies several areas for future development of the GFEA algorithm. First, the algorithm should be tested in large-scale networks to see the impact of increasing the number of nodes. Secondly, a comparison should be conducted between distributed election algorithms and the multi-criteria approach for selecting a GDSS facilitator. To further validate and refine the GFEA algorithm, real-world expert feedback is crucial. Additionally, the algorithm's adaptability to different network topologies warrants investigation. Finally, other objective methods should be tested with GFEA to see how changing the weights affects the final ranking of the DMs.

Competing Interest Statement

The authors declare that they have no known competing financial interests or personal relationships that could have influenced the work reported in this paper.

Data Availability Statement

No additional data or materials were utilized for the research described in the article.

Statement on the Ethical Use of AI Tools

"Quillbot" was used to rephrase sentences that originate from other scientific works cited in this paper. "Microsoft Copilot" was used to spot the similarities between the problematics of electing a leader node in a distributed system and electing a GDSS facilitator.

Author Contribution Roles

Author 1: Writing – original draft, Visualization, Validation, Software, Methodology, Conceptualization, Investigation, Formal analysis.

Author 2: Writing – review and editing, Methodology, Supervision, Project administration.

References

[1] R. Mahajan, P. R. Patil, M. Shahakar, and A. Potgantwar, "An analytical evaluation of various approaches for load optimization in distributed system," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 12, no. 1S, pp. 526–548, 2023.

[2] J. T. Fornerón Martínez, F. Agostini, and D. L. La Red Martínez, "Resource and process management with a decision model based on fuzzy logic," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 8, no. 2, pp. 134–149, 2023, doi: 10.9781/ijimai.2023.02.009.

[3] T. Biswas, R. Bhardwaj, A. K. Ray, and P. Kuila, "A novel leader election algorithm based on resources for ring networks," *International Journal of Communication Systems*, vol. 31, no. 10, Jul. 2018, doi: 10.1002/dac.3583.

[4] J. S. Ereifej, "Impact of group decision support system (gdss) on organizational decision making in telecommunication sector in jordan," *We'Ken-International Journal of Basic and Applied Sciences*, vol. 2, no. 2, pp. 15, Dec. 2017, doi: 10.21904/weken/2017/v2/i2/120588.

[5] G. DeSanctis and B. Gallupe, "Group decision support systems," *ACM SIGMIS Database: the DATABASE for Advances in Information Systems*, vol. 16, no. 2, pp. 3–10, Dec. 1984, doi: 10.1145/1040688.1040689.

[6] K. J. Euske and D. R. Dolk, "Control strategies for group decision support systems: An end-user computing model," *Eur J Oper Res*, vol. 46, no. 2, pp. 247–259, May 1990, doi: 10.1016/0377-2217(90)90135-X.

[7] S. French, "Web-enabled strategic GDSS, e-democracy and Arrow's theorem: A Bayesian perspective," *Decision Support Systems*, vol. 43, no. 4, pp. 1476–1484, Aug. 2007, doi: 10.1016/j.dss.2006.06.003.

[8] M. Limayem, J. E. Lee-Partridge, G. W. Dickson, and G. DeSanctis, "Enhancing GDSS effectiveness: automated versus human facilitation," in *[1993] Proceedings of the Twenty-sixth Hawaii International Conference on System Sciences*, Wailea, HI, USA: IEEE, Jan. 1993, pp. 95–101. doi: 10.1109/HICSS.1993.284171.

[9] S. Mohammedi Taieb and M. A. Laredj, "Proposition of a new tool for the election of group decision support system facilitator based on the multi-criteria approach," *STUDIES IN ENGINEERING AND EXACT SCIENCES*, vol. 5, no. 2, pp. 01–25, Oct. 2024, doi: 10.54021/seesv5n2-303.

[10] M. Keshavarz-Ghorabaee, M. Amiri, E. K. Zavadskas, Z. Turskis, and J. Antucheviciene, "Determination of objective weights using a new method based on the removal effects of criteria (MEREC)," *Symmetry (Basel)*, vol. 13, no. 4, pp. 525, Mar. 2021, doi: 10.3390/sym13040525.

[11] L. Sperling and S. S. Kulkarni, "Privacy-preserving methods for outlier-resistant average consensus and shallow ranked vote leader election," Jan. 2023, doi: https://doi.org/10.48550/arXiv.2301.11882.

[12] F. Jiang, Y. Cheng, C. Dong, and E. Yu, "A novel weight-based leader election approach for split brain in distributed system," in *IOP Conference Series: Materials Science and Engineering*, Wuhan, China: Institute of Physics Publishing, Nov. 2020, pp. 012005. doi: 10.1088/1757-899X/719/1/012005.

[13] Y. Luo, Y. Chen, Q. Chen, and Q. Liang, "A new election algorithm for DPos consensus mechanism in blockchain," in *Proceedings - 7th International Conference on Digital Home, ICDH 2018*, Guilin, China: Institute of Electrical and Electronics Engineers Inc., Feb. 2019, pp. 116–120. doi: 10.1109/ICDH.2018.00029.

[14] M. A. Haddar, "SEALEA: scalable and energy aware K-leaders election algorithm in IOT wireless sensor networks," *Wireless Personal Communications*, vol. 125, no. 1, pp. 209–229, Jul. 2022, doi: 10.1007/s11277-022-09547-8.

[15] H. C. Cahng and C. C. Lo, "A consensus-based leader election algorithm for wireless ad hoc networks," in *Proceedings - 2012 International Symposium on Computer, Consumer and Control, IS3C 2012*, Taichung, Taiwan: IEEE, Jun. 2012, pp. 232–235. doi: 10.1109/IS3C.2012.66.

[16] V. Raychoudhury, J. Cao, R. Niyogi, W. Wu, and Y. Lai, "Top K-leader election in mobile ad hoc networks,"

Pervasive and Mobile Computing, vol. 13, no. 1, pp. 181–202, 2014, doi: 10.1016/j.pmcj.2013.10.003.

[17] N. Elsakaan and K. Amroun, "Distributed and reliable leader election framework for wireless sensor network (DRLEF)," in *Lecture Notes in Networks and Systems*, vol. 378 LNNS, Springer Science and Business Media Deutschland GmbH, 2022, pp. 123–141. doi: 10.1007/978-3-030-95918-0_13.

[18] A. Julian and J. Marian Jose, "Multi-criteria leader selection in ad hoc networks using fuzzy analytical hierarchy process," in *Lecture Notes in Electrical Engineering*, vol. 700, Springer, Singapore, 2021, pp. 2875–2885. doi: 10.1007/978-981-15-8221-9_269.

[19] N. Kadjouh *et al.*, "A dominating tree based leader election algorithm for smart cities IoT infrastructure," *Mobile Networks and Applications*, vol. 28, no. 2, pp. 718–731, Apr. 2023, doi: 10.1007/s11036-020-01599-z.

[20] A. Favier, L. Arantes, J. Lejeune, and P. Sens, "Centrality-based eventual leader election in dynamic networks," in *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*, Boston, MA, USA: IEEE, Nov. 2021, pp. 1–8. doi: 10.1109/NCA53618.2021.9685390.

[21] A. Biswas, A. K. Maurya, A. K. Tripathi, and S. Aknine, "FRLLE: a failure rate and load-based leader election algorithm for a bidirectional ring in distributed systems," the *Journal of Supercomputing*, vol. 77, no. 1, pp. 751–779, Jan. 2021, doi: 10.1007/s11227-020-03286-y.

[22] L. A. Rodrigues, A. E. S. Freitas, E. P. Duarte Jr., and V. Fulber-Garcia, "A hierarchical adaptive leader election algorithm for crash-recovery distributed systems," in *Proceedings of the 13th Latin-American Symposium on Dependable and Secure Computing*, New York, NY, USA: ACM, Nov. 2024, pp. 136–145. doi: 10.1145/3697090.3697102.

[23] A. Biswas, M. Singh, G. Baranwal, A. K. Tripathi, and S. Aknine, "Multi-attribute-based self-stabilizing algorithm for leader election in distributed systems," the *Journal of Supercomputing*, vol. 81, no. 4, pp. 556, Feb. 2025, doi: 10.1007/s11227-025-07043-x.

[24] G. Le Lann, "Distributed systems-towards a formal approach," *IFIP Congress*, vol. 7, pp. 155–160, 1977.

[25] E. Chang and R. Roberts, "An improved algorithm for decentralized extrema-finding in circular configurations of processes," *Communications of the ACM*, vol. 22, no. 5, pp. 281–283, May 1979, doi: 10.1145/359104.359108.

[26] D. S. Hirschberg and J. B. Sinclair, "Decentralized extrema-finding in circular configurations of processors," *Communications of the ACM*, vol. 23, no. 11, pp. 627–628, Nov. 1980, doi: 10.1145/359024.359029.

[27] Garcia-Molina, "Elections in a distributed computing system," *IEEE Transactions on Computers*, vol. C–31, no. 1, pp. 48–59, Jan. 1982, doi: 10.1109/TC.1982.1675885.

[28] A. Laredj, B. Rouba, and C. Duvallet, "Multi-criteria decision aid for group facilitator election: Application to a collaborative e-maintenance process," *International Journal of Decision Support System Technology*, vol. 11, no. 1, pp. 93–102, Jan. 2019, doi: 10.4018/IJDSST.2019010105.

[29] A. Salo, R. P. Hämäläinen, and T. J. Lahtinen, "Multicriteria methods for group decision processes: an overview," in *Handbook of Group Decision and Negotiation*, Cham: Springer, Cham, 2021, pp. 863–891. doi: 10.1007/978-3-030-49629-6_16.

[30] A. Adla, P. Zarate, and J. L. Soubie, "A proposal of toolkit for GDSS facilitators," *Group Decision and Negotiation*, vol. 20, no. 1, pp. 57–77, Jan. 2011, doi: 10.1007/s10726-010-9204-8.

[31] A. Ristono, P. -, P. B. Santoso, and I. P. Tama, "A literature review of criteria selection in supplier," *Journal of Industrial Engineering and Management*, vol. 11, no. 4, pp. 680, Oct. 2018, doi: 10.3926/jiem.2203.

[32] E. Y. Wirawan and R. Jayadi, "Business study of network provider development in XYZ industry area with NNI modeling (Network to Network Interface) as a stage towards smart industrial park 2021," *Journal of Theoretical and Applied Information Technology*, vol. 99, no. 6, pp. 1361–1372, Mar. 2021.

[33] N. H. Zardari, K. Ahmed, S. M. Shirazi, and Z. Bin Yusop, "Weighting methods and their effects on multi-criteria decision making model outcomes in water resources management," in SpringerBriefs in Water Science and Technology. Cham: Springer Cham, 2015. doi: 10.1007/978-3-319-12586-2.

[34] G. O. Odu, "Weighting methods for multi-criteria decision making technique," *Journal of Applied Sciences and Environmental Management*, vol. 23, no. 8, pp. 1449, Sep. 2019, doi: 10.4314/jasem.v23i8.7.

[35] F. Ecer and E. Aycin, "Novel comprehensive MEREC weighting-based score aggregation model for measuring innovation performance: the case of G7 countries," *Informatica*, vol. 34, no. 1, pp. 53–83, Sep. 2023, doi: 10.15388/22-INFOR494.

[36] K. Choumas and T. Korakis, "On using raft over networks: improving leader election," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1129–1141, Jun. 2022, doi: 10.1109/TNSM.2022.3147958.

[37] D. Hedjazi, "Constructing collective competence: a new CSCW-based approach," *International Journal of Information and Communication Technology*, vol. 12, no. 3/4, pp. 393, 2018, doi: 10.1504/IJICT.2018.090418.

[38] M. A. Laredj and K. Bouamrane, "Workflow specification for interaction management between experts in a cooperative remote diagnosis process," *Computer Science and Information Systems*, vol. 8, no. 3, pp. 573–590, 2011, doi: 10.2298/csis1003260011.

[39] A. Kovacevic, O. Heckmann, N. C. Liebau, and R. Steinmetz, "Location awareness-improving distributed multimedia communication," *Proceedings of the IEEE*, vol. 96, no. 1, pp. 131–142, 2008, doi: 10.1109/JPROC.2007.909913.