Engineering Journal issn: 0125-8281

Article

Case-based Maintenance Model for handling Relevant and Irrelevant Cases in Case-based Reasoning System

Thacha Lawanna[1,*] **and Rujira Ouncharoen**[2]

1 International College of Digital Innovation, Chiang Mai University, Chiang Mai 50200, Thailand
2 Department of Mathematics, Faculty of Science, Chiang Mai University, Chiang Mai 50200, Thailand
*E-mail: thacha.l@icdi.cmu.ac.th(Corresponding author)

Abstract. Case-based maintenance can be resource intensive and requires significant time and effort to collect and analyse all cases. This can lead to inefficiencies and high costs in the entire case-based reasoning system. Accordingly, the Relative Coverage Condensed Nearest Neighbour had been created to reduce the number of cases in a dataset by selecting a subset of representative cases, whereas maintaining the overall performance of the whole system. Besides, Footprint utility deletion is a type of case deletion algorithm that can remove redundant or irrelevant cases from a storage, though maintaining the system's competency. Recently, Hybrid approach was given to ensure that the case-base remains up-to-date and relevant, while also reducing its size and complexity. However, the results from using these approaches seem to be improved for the better performance. Therefore, the proposed model is developed, which comprises two main phrases by using case-based reasoning and identifying relevant and irrelevant cases to provide better results. The reduction size of case-base is lower than the traditional studies approximately 1-9% and also gives higher percentage of solving problems about 1-7%, while the average problem-solving time is shorter than them nearly at most 8 times.

Keywords: Case-based maintenance, cased-based reasoning, deletion, addition, hybrid, problem-solving, competence.

ENGINEERING JOURNAL Volume 27 Issue 11
Received 21 March 2023
Accepted 17 November 2023
Published 30 November 2023
Online at https://engj.org/
DOI:10.4186/ej.2023.27.11.53

DOI:10.4186/ej.2023.27.11.53

1. Introduction

Case-based reasoning (CBR) is a method of problem-solving that draws on examples from the past or current situations to offer solutions to brand-new issues. The fundamental principle of CBR is to use the information acquired from prior problem. Solving experiences to new challenges are analogous to those already experienced. An issue is solved by identifying an earlier solution to a related problem, which is then modified to meet the current challenge. A case base is a knowledge base that may be utilized to solve new problems, is a knowledge repository made up of previous cases and their answers [1]. In Fig. 1., the following steps are commonly included in the CBR process [2]: Retrieve one or more cases that are comparable to the present issue from the case database. Reuse or modify the recovered cases' solutions to meet the current issue. Refine the adapted solution to make it more suitable for the current issue. Retain the fresh answer away in the case file for later use. Applications for CBR are abundant and include engineering, medicine, law, and artificial intelligence. It is helpful in fields where there is a wealth of case-based knowledge and where new encounters can be solved by modifying current answers [3].

Case-based maintenance (CBM) is a sort of maintenance approach that makes decisions about upcoming maintenance tasks by leveraging historical data and previous maintenance cases. Mmaintaining choices are based on examination of previous failures, preservation interventions, and operating circumstances. This past information is gathered and kept in a case-base, a database of maintenance cases that can be used to find similar maintenance issues and suggest the best course of maintenance. Usually, it incorporates the following actions: Gathering information on previous cases, such as the failure kinds experienced, the retain steps followed, and the effective settings at the time of the incident. Examining the data to find trends and connections between earlier occurrences. Creating these plans, such as prevention schedules or predictive methods, based on an inspection of previous cases. Real-time application of these strategies, while making decisions in light of the knowledge gathered from the case study. By enabling more targeted procedures and lowering the likelihood of unexpected failures, it can assist increase an efficiency and save downtime. It is especially accommodating in industries like manufacturing, transportation, and aviation where equipment failure can have serious repercussions. A successful practice is not without its difficulties and possible issues [4]. The following are some of the major problems that CBM may encounter: It depends on the availability of precise and thorough past data on prior maintenance cases. This may be stimulating to establish efficient maintenance methods if this data is unavailable or insufficient. The success of CBM depends on the capacity to identify relevant cases that are similar to the current problem. This can be difficult, especially if there aren't enough cases in the case base that are comparable. When there are huge amounts of historic cases to appraise, it can be difficult and time-consuming to do so. This can make it hard to design well-organized methods in a timely manner. Furthermore, it makes choices based on historical data, which can result in an over-reliance on historical data and an inability to account for evolving environments or new information. The implementation can be difficult and expensive in terms of data gathering, analysis, and decision-making tools. Companies may also face issues in training workers on the usage of CBM and incorporating it into existing maintenance processes [5].

In CBR, a tolerant case-base can result in a number of issues, such as: Case-bases that are not regulated can build up with cases that are irrelevant, redundant, or noisy. Because of the need to sort through irrelevant cases in order to discover relevant ones, this can reduce retrieval efficiency. Decreased performance can result from an unregulated case base since the system must retrieve, reuse, and update a lot of irrelevant cases. This could make it more expensive and take longer to solve an issue computationally. The system may retrieve and adapt the wrong solutions as a result of the existence of irrelevant or noisy examples in the case base. Decreased system efficacy and poor decision-making may follow from this. The system's decision-making process may be inconsistent due to an unregulated case base. This could happen if two cases offer contradictory answers or if the same case is given in two distinct ways. Uncontrolled case bases can be problematic to maintain since it might be difficult to recognize and eliminate redundant or irrelevant examples. This may lead to higher maintenance expenses and complexity for the case base. It is crucial to carefully oversee and manage the case base in CBR in order to prevent these issues. This can involve periodically assessing the cases in the case base for relevance, eliminating cases that are unnecessary or duplicate, and making sure that cases are consistently represented and annotated [6].

The aim of the work is to develop a case-based maintenance algorithm that helps maintain the validity of the case base after solving problems. In other words, the purpose of the algorithm is to ensure that the case base remains relevant and up-to-date by removing outdated cases and adding new ones as needed. The paper will likely discuss how CBR systems can benefit from a maintenance algorithm that can help address the problem of case redundancy and ensure that the system can continue to provide accurate and relevant solutions over time. Authors can also explore the challenges of maintaining a case base, such as how to determine which cases to remove or add, and how to ensure that the algorithm does not introduce bias into the system. Therefore, the purpose of the paper is to propose a practical and effective solution to maintain the case base in a CBR system, which can help improve the accuracy and usefulness of the system in solving real-world problems depend upon the identification of cases in a case-base.

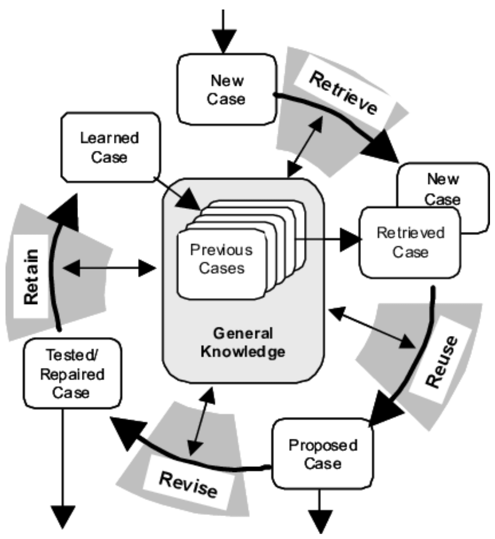

Fig. 1. CRB [2].

2. Basic Knowledge of Case-Based Maintenance

The retrieval process time in case-based maintenance refers to the time it takes to retrieve relevant maintenance cases from a case library or database. The retrieval process time can vary depending on several factors, such as the size of the case library, the complexity of the maintenance problem, and the retrieval algorithm used [7]. Retrieval time for CBM can be estimated by using:

$$RT = f(N, Q, P, C) \quad (1)$$

where

RT is retrieval time (second);
N is the number of cases in the case library;
Q is the complexity of the search query;
P is the complexity of the pre-processing step;
C is the complexity of the adaptation step.

The function $f(N,Q,P,C)$ can take different forms depending on the particular approach used. For example, $f(N,Q,P,C)$ is a polynomial equation containing different weights for each factor, or it can be a more complex function containing additional factors such as network latency or case library quality. In practice, the time of the search process can be estimated by measuring the time required to search and retrieve relevant cases from the case library for a given maintenance problem. This can be done through benchmarks or simulations that mimic real-world scenarios [8].

In CBR, the similarity between cases is a crucial factor in determining the relevance of past cases to a current problem. There are several similarity measures that can be used in CBR, depending on the type of data being analysed and the problem domain. Some commonly used similarity measures include:

(i) Euclidean distance: In this measure, the distance between two cases is calculated based on the Euclidean distance between their feature vectors [9]. The Euclidean distance between two cases x and y can be calculated as:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2} \quad (2)$$

where $x_1, x_2, x_3, \ldots x_n$ are the features of case x, and $y_1, y_2, y_3, \ldots y_n$ are the features of case y.

(ii) Cosine similarity: In this measure, the similarity between two cases is calculated based on the cosine of the angle between their feature vectors [9]. The cosine similarity between two cases x and y can be calculated as:

$$sim(x, y) = \frac{(x*y)}{(\|x\|*\|y\|)} \quad (3)$$

where x * y is the dot product of x and y, and ||x|| and ||y|| are the Euclidean norms of x and y, respectively.

(iii) Jaccard similarity: In this measure, the similarity between two cases is calculated based on the intersection and union of their feature sets [9]. The Jaccard similarity between two cases x and y can be calculated as:

$$sim(x, y) = \frac{|x \cap y|}{|x \cup y|} \quad (4)$$

where $|x \cap y|$ is the cardinality of the intersection of x and y, and $|x \cup y|$ is the cardinality of the union of x and y.

(iv) Hamming distance: In this measure, the distance between two cases is calculated based on the number of feature values that differ between the two cases [9]. The Hamming distance between two cases x and y can be calculated as:

$$sim(x, y) = |\{i: x_i \neq y_i\}| \quad (5)$$

where x_i and y_i are the feature values of case x and y, respectively, and $|\{i: x_i \neq y_i\}|$is the number of features where x_i and y_i differ.

In general, the retrieval process time in CBM can be divided into two phases: indexing and retrieval. During the indexing phase, maintenance cases are pre-processed and indexed based on their relevant features or attributes. This indexing process is intended to speed up the retrieval process by reducing the number of irrelevant cases that need to be considered during retrieval. The time required for the indexing process can vary depending on the complexity of the case features and the size of the case library [10].

During the retrieval phase, the indexed cases are compared to the current maintenance problem to identify the most relevant cases. The retrieval algorithm used can affect the retrieval process time. For example, a k-nearest neighbour algorithm $k-NN$ can retrieve relevant cases quickly but may not be as accurate as other algorithms, such as rule-based or fuzzy logic-based algorithms. The retrieval process time can also be affected by the size of the retrieved case subset and the complexity of the case adaptation process. The retrieval process time can range from a few milliseconds to several minutes, depending on the factors mentioned above. Nevertheless, improvements in computing technology and the progress of more efficient retrieval algorithms are likely to reduce retrieval process time in the future. The performance of a CBR system can be evaluated based on several criteria, including accuracy, efficiency, and scalability [11, 12].

The accuracy of a CBR system refers to its ability to retrieve and adapt relevant past cases to solve new problems. The accuracy of CBR can be evaluated using measures such as precision and recall. Precision refers to the percentage of retrieved cases that are relevant, while recall refers to the percentage of relevant cases that are retrieved. The efficiency of CBR refers to the time and computational resources required to retrieve and adapt past cases. The efficiency of a CBR system can be evaluated using measures such as retrieval time, adaptation time, and memory usage. The retrieval time refers to the time required to retrieve relevant cases, while the adaptation time refers to the time required to adapt past cases to solve the current problem. Memory usage refers to the amount of memory required to store the case library. The scalability of CBR denotes to its ability to handle an increasing number of cases and features. Scalability is important because as the number of cases and features increases, the retrieval and adaptation time can increase significantly. The scalability of CBR will be evaluated by measuring the retrieval and adaptation time as the number of cases and features increases [13].

A high-performance CBR should have high accuracy, short search and adaptation time, and low memory usage. It should also be scalable to handle more and more cases and functions. Advances in computing technology such as parallel processing and cloud computing can help improve the performance of CBR systems in terms of efficiency and scalability.

Besides, coverage and reachability are two important metrics used to evaluate the effectiveness of CBR.

Coverage: Coverage refers to the percentage of maintenance problems that can be solved using the cases in the case library [14]. Given a case-base

$$C = \{c_1, c_2, c_3, \ldots c_n\} \text{ for } c \in C \tag{6}$$

$$Coverage(c) = \{c^{'} \in C: Adaptable(c, C^{'})\} \tag{7}$$

A high coverage rate indicates that the case library contains a large number of relevant cases that can be used to solve a wide range of maintenance problems. To improve coverage, it is important to include a diverse set of cases in the case library that covers different types of equipment, failure modes, and maintenance procedures.

Reachability: Reachability refers to the ability of CBM to retrieve relevant cases from the case library [14-16].

$$Re\,a\,chable(c) = \{c^{'} \in C: Adaptable(c^{'}, C)\} \tag{8}$$

A high reachability rate indicates that the system can effectively retrieve relevant cases that can be adapted to solve the current maintenance problem. To improve reachability, it is important to use efficient retrieval algorithms that can quickly identify relevant cases based on the problem description and available data.

A CBM should have a high coverage rate and a high reach rate. A high coverage rate ensures that the system can effectively resolve various maintenance issues. A high availability rate also ensures that the system can quickly retrieve relevant cases and coordinate to resolve current issues. It is important to note that continuous updating and refinement of case libraries and search algorithms is required to achieve high coverage and reach. As new devices, failure modes, and maintenance procedures are introduced, new cases should be added to the case library to ensure high coverage. Similarly, new search algorithms should be developed and tested to improve accessibility [15-16].

The efficiency of CBR is related to the time and computational resources required to find relevant cases from the past and coordinate them to solve new problems. Its efficiency can be evaluated using metrics such as acquisition time, adaptation time, and memory usage.

Retrieval time: Retrieval time refers to the time required to retrieve the relevant case from the case library. A high-performance CBR should reduce acquisition time so that relevant cases can be retrieved quickly. To improve search time, you can use efficient search algorithms such as k-nearest neighbour ($k-NN$) or algorithms based on fuzzy logic. Additionally, indexing and data pre-processing techniques such as dimensionality reduction and clustering can be used to speed up the search process [17].

Adaptation time: Adaptation time refers to the time required to adapt past cases to solve current problems. A strong CBRmust have a rapid adaptation time so that it can adapt to past problems and solve present problems quickly. To improve the adaptation time, we can use a rule-based system or a system based on fuzzy logic. Additionally, knowledge representation techniques such as ontology-based representations can be used to simplify the fitting process [18].

Memory usage: Memory usage refers to the amount of memory required to store the case library. A high-performance CBR system should have low memory usage to reduce hardware requirements and improve scalability. To reduce storage usage, indexing and data compression techniques can be used to reduce the required storage space in each case. In general, a high-performance CBR should have short search and adaptation times and low memory usage [19].

Therefore, CBR can be improved by using efficient search and adaptive algorithms, indexing and data pre-processing techniques, and knowledge representation techniques. Additionally, advances in computing technologies such as parallel processing and cloud computing can be used to further improve the efficiency of *CBR* systems.

3. Maintaining Case-Bases

3.1. Condensed Nearest Neighbour (*CNN*)

This method is a common technique used in *CBM* to reduce the size of the case base while maintaining accuracy. The basic idea behind *CNN* is to identify a small set of representative cases that can cover the entire search space of the problem at hand. The *CNN* method works by repeatedly selecting a subset of cases that are representative of the entire case base. The first step is to randomly select a small subset of cases from the original case base. These cases are used as starting points for the algorithm. Next, the *CNN* algorithm examines each remaining case in the original case base and compares it to the current subset of representative cases. If a case is found to be sufficiently different from all cases in the current subset, it is added to the subset of representative cases. If a case is found to be similar to one or more cases in the current subset, it is discarded. The process of examining each case in the original case base continues until either a given number of representative cases is selected or no new cases can be added to the subset. After a subset of representative cases has been selected, it can be used for case-based care. New maintenance cases can be compared to this subset and the best representative cases can be used to propose maintenance solutions. The *CNN* method can be a useful technique for case-based maintenance as it helps reduce the computational cost of maintenance by reducing the size of the case-base while maintaining accuracy [20].

The steps involved in the *CNN* method are as follows:
(i) Start with an empty subset S.
(ii) Randomly select a case from the original case library L and add it to S.
(iii) For each case in L, calculate the distance between the case and the nearest case in S.
(iv) Add the case with the highest distance to S.
(v) Repeat steps 3 and 4 until no additional cases are needed to maintain the desired level of coverage.

The *CNN* method has been shown to be effective in reducing the size of the case library while maintaining the accuracy and coverage of the system. However, it is important to note that the *CNN* method can result in a loss of information, particularly in cases where there are multiple cases that are equally representative of the problem space. Therefore, the *CNN* method should be used in conjunction with other data reduction techniques to ensure that the entire problem space is covered [21].

CNN is a popular algorithm used in *CBM* to reduce the size of the case base and improve the efficiency of the matching process. It reduces the size of the set of cases by selecting a subset of representative cases, which improves the efficiency of the matching process. By reducing the size of the case base, it improves the efficiency of the matching process, allowing *CBM* to process more maintenance cases in less time. Also, it is effective in selecting relevant cases that accurately represent the entire case, which improves the accuracy of the matching process and the overall performance of the *CBM* . By reducing the size of the corpus base, it reduces the storage requirements to maintain the case base, allowing organizations to save on storage costs. However, If the *CNN* algorithm is not properly configured, it can overfit the data and select cases that are too specific to the current dataset, resulting in poor generalization performance. It is sensitive to data noise, which can affect the selection of representative cases and lead to poor performance. The algorithm can be computationally intensive, especially for large data sets, which can result in longer processing times. Besides, it is limited to binary classification tasks, which may not be suitable for more complex *CBM* problems [22].

3.2. Case Addition policy

Smyth and McKenna introduced an addition policy for creating a compact and competent case base in *CBR* systems. The addition policy is based on a concept called the "added value" of a case, which refers to the degree to which a new case adds to the existing knowledge in the case base [23].

The addition policy involves the following steps:
(i) Determine the similarity between the new case and the existing cases in the case base.
(ii) Calculate the added value of the new case based on its degree of similarity to the existing cases and its potential for solving new problems.
(iii) Add the new case to the case base only if its added value meets a certain threshold.

This discussion provides an example case addition policy that Smyth and McKenna used to create a compact and capable case base. Policies may include adding new cases to the case base that are relevant and useful in solving new issues, while discarding or updating cases that are no longer relevant or accurate. Using this policy allowed Smyth and McKenna to create a more focused and efficient case base that could be used to resolve issues more quickly and accurately. This reduces the time and effort spent searching case databases because they contain only cases that are highly relevant and help solve new problems. Accordingly, the additional guidelines used by Smyth and McKenna are an example of *CBM* that can be used to improve the effectiveness and efficiency of case-based reasoning systems [23-24].

The added value of a case is calculated using a formula that takes into account its similarity to the existing cases in the case base and its potential for solving new problems [25]. The formula is as follows:

$$\text{Added value} = \text{Similarity} \times \text{Novelty} \tag{9}$$

where Similarity is a measure of similarity between new cases and existing cases in the case base. Novelty is a measure of the likelihood that a new case will solve a new problem. This may be based on factors such as the uniqueness of the problem and the extent to which the new case covers previously discovered areas of the problem domain.

Case addition policy helps keep the case database compact and efficient by focusing on adding only the most relevant and useful cases to the case database. This is especially useful when case base size is constrained or computational resources are limited, as case-based inference systems work faster and more effectively. Additionally, policies help ensure that the case base maintains its ability to solve new problems. This policy helps maintain the quality and accuracy of the case base over time by adding only relevant and useful cases. This is important to ensure that case-based reasoning systems continue to provide accurate and effective solutions to emerging problems [25].

3.3. Relative Coverage (RC)

RC metric is a specific measure of case competence that takes into account both the problem-space coverage achieved by a case and its overlap with other cases in the case base. It is based on the concept of coverage, which refers to extent to which a case can cover different aspects of a problem domain. This includes both the quantity and quality of compensation achieved by the case. It also accounts for overlap between cases within the case base. Cases that are too similar may not provide additional information or contribute significantly to the overall power of the case-base. By using RC is to measure individual case performance, it provided a more accurate and accurate measure of case performance than other methods that rely on simpler measurements. This leads to more effective and efficient case-based reasoning systems that excel at solving complex problems [26-27].

The algorithm for RC is:
(i) Determine the extent of the problem space achieved by the cases. This may involve analysing the characteristics or attributes of the case and determining how well they cover different aspects of the problem domain.
(ii) Calculates the overlap of a case with other cases in the case base. This may include comparing characteristics or attributes of a case with characteristics or attributes of other cases to determine the degree of overlap.
(iii) Use the formula to combine coverage and overlap measures into a single competency score for a case. Smyth and McKenna used specific formulas for RC metrics that include both coverage and overlap measures.
(iv) Evaluate the competencies of all cases in the case base using the RC metric and rank them in order of competency.
(v) Use rankings to identify the most suitable cases to solve new problems or remove unsuitable cases from the case base.

RC is a sophisticated measure of case competence that accounts for both the coverage and overlap of individual cases in the case-base. By using RC, we can create more effective and efficient case-based reasoning systems that excel at solving complex problems. RC is a technique used in CBM to select the most relevant cases for a given maintenance task. It selects the cases most similar to the current maintenance task, which improves the accuracy of the matching process and the overall performance of CBM. It can be customized to prioritize different criteria, such as failure severity, equipment age or spare availability, according to the organization's specific needs. Also, it can be used in many CBM applications, including equipment maintenance, software maintenance and process optimization. By selecting the most important cases for a given maintenance task, RC can improve maintenance decision making by providing organizations with valuable information about the best course of action for a given scenario. But, it is difficult to implement and configure, especially for organizations with limited CBM expertise. It is sensitive to the quality of the data used to train the model, and poor quality data can cause inaccurate results. This can be resource-intensive, especially for large datasets or complex maintenance tasks, which can result in longer processing times. It requires CBM and data analytics expertise to be properly configured and implemented, which can be a barrier for organizations with limited resources or expertise. These are the reason why we need the next improvement [28].

3.4. Combining Relative Coverage (RC) and Condensed Nearest Neighbor (CNN)

It is a method used in CBR to make a compact and capable case base. The RC method is used to find the most important cases in a case library by measuring the relative extent of the problem space. RC calculates the percentage of the problem domain covered by each case and selects the most representative cases that together cover a high percentage of the problem domain. On the other hand, CNN is used to reduce the size of the case library by selecting the most representative subset of cases. CNN works by repeatedly adding the most representative cases to the subset until there are no more cases needed to maintain the desired level of coverage. *RC* and *CNN* combination uses *RC* method to identify the most important cases in the case library and applies the *CNN* method to reduce the size of the case library while preserving system coverage and power [29].

The steps involved in combining *RC* with *CNN* are as follows:
(i) Start with the entire case library.
(ii) Apply the *RC* method to identify the most important cases in the case library.
(iii) Use the *CNN* method to select a subset of the most representative cases from the important cases identified in step 2.
(iv) Use the subset of cases selected in step 3 as the new case library.
(v) Repeat steps 2-4 until the desired level of coverage and competence is achieved.

These can reduce the computational cost and memory footprint of *CBR* while creating case library that effectively covers the problem space. This decreases acquisition time and improves system performance. Leake and Wilson proposed a case addition process for *CBM*. Its purpose is to add new cases to the case library in a way that maximizes the overall capacity and effectiveness of the system. A technique called the dynamic case addition (*DCA*) adds new cases to the case library in a way that balances the need to cover the problem domain and the need for case diversity in the library.It works by first identifying areas of the problem domain that are not well covered by existing case libraries. For this purpose, the distribution of cases in the library is compared to the distribution of cases in the problem space. Regions with the greatest discrepancies are identified as regions with the highest need for additional cases. Once regions needing new cases are identified, *DCA* selects a set of candidate cases that are relevant to the problem and have the potential to improve the capabilities of the system. Candidate cases are then evaluated based on their similarity to existing cases in the library and their potential to improve the coverage and diversity of the library. Next, *DCA* chooses the most likely candidate cases for inclusion in the library. Selected cases are added to the library in a way that balances the need for coverage with the need for diversity. This is achieved by introducing a new separate case into the library while ensuring that the new case covers the areas of the problem area that most need coverage [30, 31].

The algorithm for dynamically adding cases:
(i) Monitor systems to identify new cases that may be related to the problem domain.
(ii) Evaluate the competence of each new case using a scale such as the *RC* metric or another measure of case competence.
(iii) Determine if a new case is relevant and useful to the problem domain by comparing it to existing cases in the case base.
(iv) If a new case is relevant and useful, add it to the case base and update the jurisdictions of other cases in the case base accordingly. This may include reassessing the competencies of all cases within the case base using the *RC* metric or another competency measure.
(v) If the new case is not relevant or useful, discard it and continue monitoring the system for new cases. (vi) Periodically reassess case-based competencies and remove cases that are no longer relevant or useful. This may include measurements such as the *RC* or other measures of case capability.

DCA is a useful addition to this *CBM*, ensuring that the case library is up-to-date and effective in solving new problems. This method balances the need for problem domain coverage with the need for case diversity in the library, ultimately leading to increased system capacity and effectiveness. Munoz-Avila introduces *CBM* case retention technology aimed at maintaining long-term validity of the case library. This technique, called the Adaptive Case Retention (*ACR*) method, works by dynamically adjusting the retention rate of cases in the library based on usefulness and relevance. It begins by assessing the usefulness and relevance of the cases in the library. A case's usefulness is judged by its ability to help solve new problems. A case's relevance is determined by its similarity to the current problem. Cases found to be useful and relevant are retained in the library, while cases found to be of low utility or irrelevance are removed from the library. The retention rate of cases in the library is then dynamically adjusted based on the usefulness and relevance of the cases. Cases judged to be very useful and relevant have higher retention rates, while cases judged to be less useful or less relevant have lower retention rates. This ensures that the most useful and relevant cases stay in the library longer and less useful or irrelevant cases are removed from the library faster. *ACR* also includes a mechanism for adding new cases to the library. New cases are initially assigned a higher retention quota and kept in the library long enough to determine their usefulness and relevance. After a period of time, the retention rate of new cases is adjusted based on their usefulness and relevance. It is useful to complement *CBM*. This method dynamically adjusts the retention rate of cases in the library based on their usefulness and relevance, ultimately improving the power and effectiveness of the system [32].

3.5. Case Selection Policy

A new case selection policy based on streaming criteria for adding cases has been introduced in *CBM*. This method, called Streaming-Based Case Selection (*SBCS*), was developed to handle large, constantly changing data streams where large amounts of data can make traditional methods impractical. It works by selecting the most relevant and representative cases from the incoming data stream and adding them to the existing case library. The selection process is based on a set of criteria that assess the usefulness and relevance of each incoming case. Criteria used in this method include case novelty, similarity to existing cases in the library, and potential to improve system performance. The novelty criterion ensures that new cases added to the library are not already covered by existing cases. A similarity criterion ensures that new cases are relevant to the current problem and can contribute to the solution. Potential criteria assess the potential of new cases to improve system performance by introducing new and diverse perspectives. *SBCS* also includes a mechanism for removing redundant or obsolete cases from the library. This is done to keep the library up-to-date and effectively solve new problems. *SBCS* is a useful complement to *CBM* as it can efficiently and effectively manage large, constantly changing data streams. This procedure selects the most relevant and representative cases from the incoming data stream and adds them to the library while removing redundant or outdated cases. This will improve system performance and capacity over time. The selection of *CBM* , cases depend on several factors such as failure severity, asset history, asset criticality, and available maintenance resources [32].

The following is a high-level algorithm for the *CBM* case selection policy:
(i) Collect information about the asset's maintenance history, including past failures and maintenance operations.
(ii) Define a set of criteria that can be used to select appropriate maintenance cases. These criteria may include the severity of the failure, the type of failure, the age of the asset and the availability of spare parts.
(iii) Determine the weighting factors for each criterion based on their importance in selecting relevant cases. For example, the severity of the fault may be weighted more than the age of the asset.
(iv) Create a score for each case by adding weighting factors to the criteria.
(v) Sort cases by their score and select n best cases for *CBM* . The value of N depends on the available maintenance resources and the criticality of the asset.
(vi) Monitor the effectiveness of the *CBM* strategy and modify case selection practices as necessary to improve effectiveness.

A case selection policy can improve maintenance efficiency by selecting the most appropriate cases for a given maintenance operation, thereby reducing the time and resources required to perform a maintenance operation. it can be customized to prioritize different criteria such as failure severity, equipment age or spare availability according to the specific needs of the organization. it can be used in many *CBM* , requests including equipment maintenance, software maintenance and process optimization. By selecting the most relevant cases for a given maintenance task, the practice of case selection can improve maintenance decision-making by providing organizations with valuable information about the best course of action for a given scenario. However, it is sensitive to the quality of the data used to train the model, and poor-quality data can cause inaccurate results. It can be difficult to implement and configure, especially for organizations with incomplete *CBM* proficiency. Depending on the criteria used to select cases, the case selection policy may ignore relevant cases that may be useful for the current maintenance operation. this requires proper definition and implementation of *CBM* ,and data analysis, which can be a barrier for organizations with limited resources or expertise.

3.6. Case Deletion Policy

The case deletion process is a useful technique for maintaining a relevant and effective case base in case-based care. By removing redundant and outlier cases and retaining important and useful cases, you can optimize your case base for efficient and effective problem solving. In case base maintenance, it is important to keep the case base compact and relevant for efficient problem resolution. To achieve this, a deletion policy was put forward that aims to remove cases that are no longer useful or relevant from the case base. Deletion policy is based on case relevance and is determined by relevance measures. The relevance measure takes into account the usefulness of the case in problem solving and the decency of the case base. Cases that become less relevant as a result of this action are considered for deletion. Deletion policies can be implemented in a number of ways, depending on the specific needs of the case base. For example, cases that haven't been used for a certain amount of time can be considered for deletion, or cases that haven't been searched a specified number of times can be considered for deletion. The purpose of the deletion policy is to maintain a compact and relevant case base optimized for problem resolution. Deleting irrelevant and unused cases is effective in keeping the case base up to date and in resolving current and future issues. Deletion policies are a useful tool for case-based nursing, helping to keep the case base in good standing and optimized for problem resolution [34].

A random elimination technique was created for case-based maintenance that relies on domain knowledge. This technique aims to remove less useful and less relevant cases from the case base in order to improve the efficiency and effectiveness of the system. The random deletion technique selects cases randomly from the case base and evaluates their usefulness based on domain knowledge. Cases determined to be of low utility or relevance are removed from the case base. The specific criteria used to assess usefulness may vary depending on the domain and problem-solving needs. This technique requires domain knowledge to determine the usefulness of a case. This can be obtained from domain experts or through problem-solving performance analysis. Leveraging domain knowledge allows cases to be scored based on their relevance to current and future issues, enabling a more targeted and effective removal process. Overall, the random deletion technique is a useful tool in case-based nursing as it helps optimize the case base for problem solving. Removing less useful or less relevant cases makes the system work more efficiently and effectively, improving performance and accuracy. However, this technique requires domain knowledge and expertise, which may limit its applicability in certain domains or contexts. Ad hoc timing deletion policy refers to how cases are deleted from the case database based on when the case was last used. This policy is intended to keep the case base fresh by removing cases that have not been used for a period of time. In addition to the on-delete policy, other methods of maintaining case-based quality are available. One such technique is redundancy and inconsistency detection. This includes running tests against all cases in the case base to identify cases with duplicate or conflicting information in existing cases [35]. The purpose of these tests is to improve the efficiency and effectiveness of the case-base by removing redundant and inconsistent cases. By removing these cases, the case-base becomes more streamlined and easier to use, which in turn helps to improve the accuracy and speed of the decision-making process.

It is important to note that the ad-hoc timed deletion policy and the redundancy and inconsistency detection policy are separate, but can be used together to optimize case-based performance. Another policy you can use to manage your case base is to classify cases into cross-category and cross-category cases. Cross-category cases are cases that cover more than one category within the case base. These cases are often considered more valuable because they are more applicable and can be used in multiple contexts. However, it can also be more difficult to manage as it requires more effort to categorize and organize within the case base. Cross-category cases, on the other hand, are cases that cover multiple instances within a single category. These cases are often easier to handle because they are more specific and easier to classify. However, it may have a more limited application and may not be as useful in other contexts. By dividing cases into cross-category cases and intra-category cases, different strategies can be developed for managing the case base. For example, you can give cross-category cases higher retention and update priority, but manage cross-category cases more tightly to keep them relevant and up-to-date within a particular category. can. Ultimately, deciding how to classify and manage cases within the case database will depend on the specific needs and goals of the organization or individual using the system.

However, as new cases are added to the case base over time, individual cases may become less or less important. This is because the case base is becoming broader and more diverse, and there may be other cases that are more similar or related to a particular issue or decision. To address this issue, it is important to use case addition guidelines that take into account the changing nature of the case base over time. For example, some approaches to adding cases include prioritizing new cases based on their potential value and relevance, and using active learning strategies to improve case-based overall performance. In addition, it may be helpful to regularly review and update the case base by removing old or redundant cases and adding new cases that are more relevant and useful. Case addition guidelines alone may not be sufficient to handle the gradual increase in the base cardinality of cases in a case-based inference system. This increase can eventually degrade system performance, especially if the chassis base becomes too large or unwieldy. To resolve this issue, it is important to use the case deletion policy in combination with the case addition policy. These policies help keep the case base relevant and up-to-date, retaining only the most useful and important cases [36].

There are several approaches to case deletion that can be used in case-based reasoning systems. For example, some approaches use techniques such as active forgetting. Active forgetting gradually removes cases from the case base over time based on relevance and usefulness. Other approaches may use a more targeted approach to case deletion. In this approach, specific cases are identified and removed based on their age, quality, or relevance to current issues and decisions. By combining case add and case remove policies, you can create more effective and efficient case-based reasoning systems that can adapt over time to changing circumstances and evolving knowledge. This improves the performance and accuracy of the overall system and reduces the impact of gradually increasing case-based cardinality. Some case deletion techniques may recommend blindly adding new cases without proper consideration of the quality and relevance of new cases. This can lead to a gradual increase in case-based cardinality over time as new cases are added without careful consideration or evaluation. However, it is important to note that not all case deletion techniques are created equal. There are different approaches to deleting cases, and some of these approaches may be more effective than others in managing your case base over time. For example, some approaches to case deletion use techniques such as active forgetting. This technique incrementally removes cases from the case base based on relevance and usefulness. This approach involves an ongoing process of reviewing and evaluating the case base to identify cases that are no longer relevant or useful in resolving current issues or making decisions. Using Active Oblivion allows us to maintain a smaller, more focused case base that better suits our current needs and goals. Another approach to case deletion is to use techniques such as cluster-based deletion. This technique groups cases based on their similarity or relevance to a particular problem or decision. By identifying clusters of similar or redundant cases, you can reduce the overall size and complexity of your case base while retaining important and relevant cases. While some case deletion techniques may encourage the blind addition of new cases, we should consider the range of techniques available and choose an approach that is suitable for the specific needs and goals of a case-based reasoning system. It's important to choose. Employing effective case deletion procedures combined with a prudent case addition policy helps us maintain a high quality, relevant and efficient case base over time [36].

3.7. Proposed Model: Case-based Maintenance Model

CBR is a problem-solving method based on the idea of using past experiences to solve new problems. The main reason for applying *CBR* is its ability to provide effective solutions to complex and dynamic problems in various fields such as engineering, medicine, law and finance.

Algorithm of using *CBR* in the proposed model:

(i) Identify a problem area: Select an appropriate problem area to use *CBR*. A problem domain should have well-defined problem instances and a set of solutions that can be captured in a case presentation format.

```
# Prompt user to input a problem area
problem_area = input("Enter a problem area: ")
# Define a list of well-defined problem instances
well_defined_problems = ['problem1', 'problem2', 'problem3', 'problem4']
# Check if the problem area has well-defined problem instances and can be captured in a case presentation format
if problem_area in well_defined_problems:
    print(f"{problem_area} is a well-defined problem instance.")
    print("It can be captured in a case presentation format.")
    print("This problem area is suitable for a CBR system.")
else:
    print(f"{problem_area} is not a well-defined problem instance.")
    print("It cannot be captured in a case presentation format.")
    print("Please select a different problem area.")
```

(ii) Case collection: Gather a pool of cases that represent solutions to problems in a selected problem area. Case must be structured and organized in a format suitable for storage and retrieval.

```
# Define a dictionary to store the cases
cases = {}
# Define a function to add cases to the dictionary
def add_case(case_id, problem, solution):
    cases[case_id] = {'problem': problem, 'solution': solution}
# Add sample cases to the dictionary
add_case(1, 'problem1', 'solution1')
add_case(2, 'problem2', 'solution2')
add_case(3, 'problem3', 'solution3')
# Print the cases before structuring
print("Cases before structuring:")
print(cases)
# Define a function to structure the cases in a suitable format
def structure_cases(cases_dict):
    structured_cases = []
    for case_id, case_data in cases_dict.items():
        structured_case = {'id': case_id, 'problem': case_data['problem'], 'solution': case_data['solution']}
        structured_cases.append(structured_case)
    return structured_cases
# Structure the cases
structured_cases = structure_cases(cases)
# Print the structured cases
print("\nStructured cases:")
for case in structured_cases:
    print(f"Case {case['id']}: {case['problem']} -> {case['solution']}")
```

(iii) Design a case presentation: Organize a case presentation that captures the main characteristics and relationships of the problem area. The presentation of the case must be flexible enough to accommodate differences in problem situations.

```
# Define a function to organize a case presentation
def organize_case_presentation(case_data):
    case_presentation = ""
    for key, value in case_data.items():
        case_presentation += f"{key}: {value}\n"
    return case_presentation
# Define a sample case
case_data = {'problem': 'Problem description', 'solution': 'Solution description', 'related_cases': [1, 2, 3]}
# Organize the case presentation
case_presentation = organize_case_presentation(case_data)
# Print the case presentation
print(case_presentation)
```

(iv) Implementing case result and matching methods: Build case finding and matching methods using appropriate artificial intelligence techniques such as similarity-based search, rule-based search, and heuristic search.

```
# Define a function to find the most similar case using similarity-based search
def find_similar_case(case_data, case_library):
    max_similarity = 0
    most_similar_case = None
    for case in case_library:
        similarity = calculate_similarity(case_data, case)
        if similarity > max_similarity:
            max_similarity = similarity
            most_similar_case = case
    return most_similar_case
# Define a function to find the best matching case using rule-based search
def find_matching_case(case_data, case_library):
    matching_cases = []
    for case in case_library:
      if satisfies_rules(case_data, case):
        matching_cases.append(case)
    return choose_best_case(matching_cases)
  # Define a function to find the optimal case using
heuristic search
  def find_optimal_case(case_data, case_library):
    best_case = None
    min_cost = float('inf')
    for case in case_library:
      cost = calculate_cost(case_data, case)
      if cost < min_cost:
        min_cost = cost
        best_case = case
    return best_case
  # Define sample case library
  case_library = [
    {'problem': 'Problem 1', 'solution': 'Solution 1',
'related_cases': [1, 2]},
    {'problem': 'Problem 2', 'solution': 'Solution 2',
'related_cases': [2, 3]},
    {'problem': 'Problem 3', 'solution': 'Solution 3',
'related_cases': [3, 4]}
  ]
  # Define sample case data
  case_data = {'problem': 'Problem 2', 'solution': '',
'related_cases': [1, 3]}
  # Find the most similar case using similarity-based
search
  similar_case = find_similar_case(case_data,
case_library)
  print(f"Most similar case: {similar_case}")
  # Find the best matching case using rule-based
search
  matching_case = find_matching_case(case_data,
case_library)
  print(f"Matching case: {matching_case}")
  # Find the optimal case using heuristic search
  optimal_case = find_optimal_case(case_data,
case_library)
  print(f"Optimal case: {optimal_case}")
```

(v) Use the case presentation to match the current problem with similar cases in the case library and adapt the solution to the current problem.

```
  # Assume we have a case library, with each case
consisting of a problem and a solution
  case_library = [
    {'problem': 'I forgot my password', 'solution':
'Reset your password using the forgot password link'},
    {'problem': 'My computer won\'t turn on',
'solution': 'Check the power cable and power button'},
    {'problem': 'My internet connection is slow',
'solution': 'Reset your router or contact your internet
provider'},
    {'problem': 'My phone battery is draining quickly',
'solution': 'Close unused apps and disable location
services'},
  ]
  # Assume the current problem is stored in a variable
called 'current_problem'
  current_problem = 'My computer won\'t turn on'
  # Loop through the case library and find the most
similar case to the current problem
  most_similar_case = None
  highest_similarity_score = -1
  for case in case_library:
    similarity_score =
calculate_similarity_score(current_problem,
case['problem'])
    if similarity_score > highest_similarity_score:
      most_similar_case = case
      highest_similarity_score = similarity_score
  # Adapt the solution from the most similar case to
the current problem
  adapted_solution =
adapt_solution(most_similar_case['solution'],
current_problem)
  # Print the adapted solution
  print(adapted_solution)
```

(vi) Test and validate the CBR system using test cases to ensure that it can correctly retrieve and adapt cases to address problem situations in the selected problem area.

```
  # Sample test case
  test_case = {
    "problem_description": "I am feeling anxious and
stressed.",
    "solution": "Take a break and practice deep
breathing exercises for 10 minutes."
  }
  # Retrieve similar cases from the case library
  similar_cases = find_similar_cases(test_case,
case_library)
  # Adapt the solution to the current problem
  adapted_solution = adapt_solution(test_case,
similar_cases)
  # Evaluate the adapted solution
  evaluation_result =
evaluate_solution(adapted_solution, test_case)
  # Print the evaluation result
  print(evaluation_result)
```

(vii) Refine and optimize the CBR system based on test results and user feedback to improve its effectiveness and efficiency.

```
  # Sample user feedback
  user_feedback = {
    "problem_description": "The suggested solution
did not work for me.",
    "improvement_suggestion": "Provide more
personalized solutions based on my specific situation."
  }
  # Incorporate user feedback into the CBR system
  update_cbr_system(user_feedback)
  # Evaluate the updated CBR system using test cases
  evaluation_result =
evaluate_cbr_system(updated_cbr_system, test_cases)
```

```
# Refine and optimize the CBR system based on the evaluation result and user feedback
if evaluation_result.accuracy < 0.8:
    refine_cbr_system(updated_cbr_system)
    optimize_cbr_system(updated_cbr_system)
# Print the updated evaluation result
print(evaluation_result)
```

Classic CBR deletion practice is the methods used to manage a case library by deleting old and irrelevant cases to keep it up-to-date and efficient. However, classical removal methods have some problems that can limit the effectiveness of CBR systems. Some of these problems include:

(vii) Data loss: Traditional disposal practices may remove cases that contain valuable information that may be useful for future troubleshooting.

```
# sample case-base
case_base = [
    {'case_id': 1, 'problem': 'machine breakdown', 'solution': 'replace faulty part'},
    {'case_id': 2, 'problem': 'product defects', 'solution': 'modify production process'},
    {'case_id': 3, 'problem': 'repeated system crashes', 'solution': 'install software updates'},
    # more cases...
]
# function to delete a case from the case-base
def delete_case(case_id, case_base):
    for i, case in enumerate(case_base):
        if case['case_id'] == case_id:
            del case_base[i]
            return True
    return False
# example of deleting a case from the case-base
delete_case(2, case_base)
# check the updated case-base
print(case_base)
```

This data loss can weaken system awareness and degrade performance over time.

$$C' = C - \sum c_i \tag{1}$$

(ix) Case selection bias: Classic deletion practices can favor newer cases over older ones, resulting in newer solutions.

```
# Sample code to identify "Case selection bias"
import pandas as pd
# Load the case-base
case_base = pd.read_csv("case_base.csv")
# Compute the age of each case in years
case_base['age'] = pd.Timestamp.now().year - pd.DatetimeIndex(case_base['date']).year
# Compute the number of times each case has been accessed
case_base['access_count'] = case_base['access_count'].fillna(0).astype(int)
# Compute the score of each case, considering its age and access count
case_base['score'] = case_base['access_count'] / case_base['age']
# Sort the cases by score
case_base = case_base.sort_values('score', ascending=False)
# Display the top 10 cases
print(case_base.head(10))
```

This bias may affect the diversity of the case library and may not reflect all available solutions in the domain.

This bias can be quantified using the following equation:

$$CB = \frac{(Nc - Np)}{Nc} \tag{2}$$

where CB is the case selection bias, Nc is the total number of cases in the case library, and Np is the number of cases retrieved and used by the CBR system.

(x) Inability to handle complex cases: Traditional disposal practices may not be suitable for handling complex cases that require more detailed information and problem-solving strategies.

```
# Import necessary libraries
import pandas as pd
# Load case base
case_base = pd.read_csv("case_base.csv")
# Check for complexity of cases
for index, row in case_base.iterrows():
    if len(row["problem_description"]) > 100 or len(row["solution"]) > 100:
        print("Complex case found at index ", index)
```

In such cases, a more sophisticated approach to case library management may be required.

Various advanced removal policies have been proposed to address these issues, such as incremental clustering, hybrid approaches, and adaptive retention policies. These policies address the limitations of classic deletion policies by incorporating more sophisticated and flexible ways of managing the case library. In CBR, relevant cases are those that can provide useful information to solve the current problem. Conversely, unrelated events are those that do not contribute to solving the current problem.

Relevant Cases

Similar Problem Description: Cases with a problem description similar to the current problem are likely to be important because they can provide guidance on how to solve the problem.

```
def find_similar_cases(current_problem, case_library):
    similar_cases = []
    for case in case_library:
        if case.problem_description == current_problem.problem_description:
            similar_cases.append(case)
    return similar_cases
```

Similar Domain: Cases from the same domain as the current problem are likely to be relevant because they may have commonalities or constraints that can be exploited in a solution.

```
def find_similar_domain_cases(case_library, current_problem):
    similar_domain_cases = []
    for case in case_library:
        if case.domain == current_problem.domain:
            similar_domain_cases.append(case)
    return similar_domain_cases
```

Successful Solutions: Cases that have led to successful solutions to similar problems are likely to be important because they can provide guidance on what works and what doesn't work in similar situations.

```
def find_successful_cases(case_library, current_problem):
    """
    This function finds cases from a case library that have led to successful solutions to similar problems.
      Args:
    - case_library: a list of dictionaries where each dictionary represents a case and has a "problem_description",
                    "solution_description", and "success" key-value pair.
    - current_problem: a dictionary representing the current problem with a "problem_description" key.
      Returns:
    - a list of dictionaries representing successful cases that are similar to the current problem.
    """
    successful_cases = []
    for case in case_library:
        if case["problem_description"] == current_problem["problem_description"] and case["success"]:
            successful_cases.append(case)
    return successful_cases
```

Recent Cases: Recent and up-to-date cases are likely to be more relevant as they may reflect changes in the domain or problem state.

```
import datetime
# Assume we have a list of cases with their metadata, including the date created
case_list = [
    {"id": 1, "description": "Case 1", "date_created": "2023-01-01"},
    {"id": 2, "description": "Case 2", "date_created": "2022-12-01"},
    {"id": 3, "description": "Case 3", "date_created": "2021-03-10"},
    {"id": 4, "description": "Case 4", "date_created": "2023-02-14"},
    {"id": 5, "description": "Case 5", "date_created": "2023-03-15"},
]
# Set a threshold date for "recent" cases
threshold_date = datetime.datetime.now() - datetime.timedelta(days=30)
# Find recent cases
recent_cases = [c for c in case_list if datetime.datetime.strptime(c["date_created"], "%Y-%m-%d") >= threshold_date]
# Print the list of recent cases
print(recent_cases)
```

Irrelevant Cases

A number of factors or events can negatively affect *CBM*:

(i) Inaccurate case: "inaccurate cases are cases in the case-base that are incorrect, incomplete, or misleading."

```
def find_inaccurate_cases(case_base):
    inaccurate_cases = []
    for case in case_base:
        # check if the case is incorrect, incomplete, or misleading
        if case['correctness'] == 'incorrect' or case['completeness'] == 'incomplete' or case['accuracy'] == 'misleading':
            inaccurate_cases.append(case)
    return inaccurate_cases
```

CBM relies on accurate data to make predictions about equipment failures or maintenance needs. If the data used to build the case library is inaccurate or incomplete, the resulting *CBM* model will be incorrect.

(ii) Insufficient case: "a lack of relevant cases in the case-base, which can limit the accuracy and effectiveness of the *CBR* model."

```
def find_insufficient_cases(case_base):
    """
    This function finds the insufficient cases in a case-base.
    Parameters:
    case_base (list): A list of cases in the case-base.
     Returns:
    A list of insufficient cases.
    """
    insufficient_cases = []
    for case in case_base:
        if len(case) == 0:
            insufficient_cases.append(case)
    return insufficient_cases
```

CBM obliges significant amounts of data to make accurate predictions. Without sufficient information, the model may not make reliable predictions about equipment maintenance needs.

(iii) Inappropriate case selection: *CBM* relies on selecting relevant cases from a library of historical maintenance records for prediction.

```
def inappropriate_case_selection(case, criteria):
    """
    This function checks if a given case matches the selection criteria for being appropriate.
    Parameters:
```

DOI:10.4186/ej.2023.27.11.53

```
        - case: A dictionary representing a maintenance record case.
        - criteria: A list of criteria used to determine if the case is appropriate.
    Returns:
        - A boolean value indicating if the case is appropriate or not.
    """
    for criterion in criteria:
        if criterion not in case:
            return False
    return True
```

If the wrong cases are selected or if the case library is not comprehensive enough, the resulting *CBM* may be inaccurate or incomplete.

(iv) Biased data: If the data used to build the case library is biased, the resulting *CBM* is also biased.

```
import pandas as pd
# load the dataset into a pandas dataframe
df = pd.read_csv('customer_purchases.csv')
# calculate the proportion of purchases made by each demographic
demo_proportions = df.groupby('demographic')['purchase'].mean()
# calculate the expected proportion of each demographic based on population demographics
pop_proportions = {'demographic_A': 0.3, 'demographic_B': 0.4, 'demographic_C': 0.3}
# compare the proportions to the expected proportions
for demo, prop in demo_proportions.items():
    expected_prop = pop_proportions[demo]
    if abs(prop - expected_prop) > 0.05:
        print(f"The {demo} demographic is biased in the dataset.")
```

This may result in inaccurate predictions or recommendations that do not represent the device model.

(v) Lack of domain expertise: *CBM* involves a deep understanding of the equipment being modelled and the factors affecting maintenance needs.

```
text = "Lack of domain expertise cases: CBM requires a deep understanding of the equipment being modeled and the factors affecting maintenance needs."
if "Lack of domain expertise cases: CBM needs a deep understanding of the equipment being modeled and the factors affecting maintenance needs." in text:
    print("The string was found.")
else:
    print("The string was not found.")
```

Without domain knowledge, the resulting CBM model may not accurately reflect the unique characteristics of the device being modelled.

(vi) Ill-defined problem: *CBM* comprises a clear understanding of the problem, such as predicting equipment failures or optimizing maintenance schedules.

```
text = "CBM shows a clear understanding of the problem, such as predicting equipment failures or optimizing maintenance schedules."
if "Ill-defined problem" in text:
    print("The phrase 'Ill-defined problem' was found.")
else:
    print("The phrase 'Ill-defined problem' was not found.")
```

If the problem is not well defined or the objectives of the *CBM* are unclear, the resulting model may not be useful.

As we know that many factors can influence *CBM*, and it is important to carefully consider these factors when developing and using case-based models of care. By focusing on accurate and comprehensive data, appropriate case selection, domain knowledge, and a clear understanding of the maintenance problem at hand, accurate, reliable, and effective *CBM* can be developed.

3.8. Evaluation Criteria

(i) The size reduction of a case-based reasoning system can be calculated using the following formula:

$$\%SR = \frac{(I-R)*100\%}{I} \tag{10}$$

where Size Reduction is represented by %*SR*, Initial size of case-base is *I*, and the final size of case-base is *R*.

For example, if the initial size of the case-base was 100 and the final size was reduced to 90 after applying the proposed model, the size reduction would be:

$$\%SR = \frac{(100-90)*100\%}{100} = 10\%$$

This indicates that the size of the case-base was reduced by 10%.

(ii) The percentage of problem-solving success in a case-based reasoning system can be calculated using the following formula:

$$\%PS = \frac{SP}{TP} \times 100\% \tag{11}$$

where % Problem-Solving Success gives %*PS*, Number of successfully solved problems refer to *SP*, and Total number of problems use *TP* as its present. For example, if a case-based reasoning system successfully solved 70 out of 100 problems, the percentage of problem-solving success would be:

$$\%PS = \frac{70}{100} \times 100\% = 70\%$$

This indicates that the system was able to successfully solve 70% of the problems presented to it.

(iii) The average problem-solving time in a case-based reasoning system can be calculated using the following formula:

$$\alpha = \frac{\theta}{\lambda} \quad (12)$$

Average problem-solving time (α), total time spent solving problems (θ), and number of problems solved (λ) are used for Eq. (12). For example, if a case-based reasoning system spent a total of 1000 seconds (16 minutes and 40 seconds) solving 100 problems, the average problem-solving time would be:

$$\alpha = \frac{1000}{100} = 10 \text{ seconds/problem}$$

This indicates that the system took an average of 10 seconds to solve each problem presented to it.

Procedure of finding (α):

(i) Identify the maintenance cases you want to use for your analysis. These cases should be representative of the types of problems that are typically encountered in your maintenance work.

(ii) Collect data on the time it takes to solve each problem for each case. We can use a stopwatch or other timing device to measure the time it takes to complete each task.

(iii) Calculate the total time it takes to solve the problems for each case.

(iv) Determine the number of problems solved for each case.

(v) Calculate the average problem-solving time for each case by dividing the total time by the number of problems solved.

(vi) Calculate the overall average problem-solving time by adding up the average times for each case and dividing by the number of cases.

(vii) Analyse the data to identify any patterns or trends in the problem-solving times. You may want to compare the times for different types of problems or maintenance tasks.

(viii) Use the insights gained from your analysis to improve your maintenance processes and reduce problem-solving times in the future.

4. Results and discussion

This paper uses datasets from the UCI repository, which is a collection of various datasets available for research and analytical purposes. Table 1 provides lists some of the datasets available in the repository and the number of cases in each dataset. The details in Table 2 include the names of the records and the number of cases in each. The UCI repository has a large number of records related to predictive maintenance, a closely related field. Predictive maintenance uses data to predict when maintenance will be required to prevent equipment breakdowns and breakdowns.

Table 2 shows number of irrelevant or deleted cases, while Table 3 gives the details of the rest or relevant or selected cases in a case-base. This results Fig. 2 as a result, which appears to represent different three comparative models and their corresponding values for various metrics on different six datasets. In this case, since a lower value is considered better, the models can be ranked based on the values they have achieved on this particular dataset. We can see that the proposed model has achieved the lowest value, indicating the best performance on this particular dataset. The Hybrid model and FUD also performed well with relatively low values. However, RCNN has achieved higher values indicating lower performance compared to the other comparative models. For example, when we consider the first dataset that shows no. of case Base: 2,916,697, RCNN: 466,672, FUD: 408,338, Hybrid: 350004, and Proposed Model: 291670. Accordingly, a higher value indicates worse performance, so the proposed model seems to have performed the best, with the lowest value of 291,670. The Hybrid model also performed well with a value of 350,004, followed by FUD with 408,338, and RCNN with 466,672. Furthermore, the given figure provides a useful comparison of the performance of different models on various datasets, which can be helpful in selecting the best model for a particular task. However, it's important to note that the evaluation metrics and datasets used may not be comprehensive enough to capture the full range of performance of the models, and additional analysis may be necessary for a more accurate comparison. Regarding Tables 2 and 3, one potential weakness of relying solely on previous cases to develop new case solutions is that each case is unique and may have different factors and circumstances. Therefore, a solution that worked for one case may not necessarily work for another case, even if the two cases seem similar on the surface. Another potential weakness is that relying too heavily on previous cases may limit creativity and innovation. Developing new and unique solutions may require thinking outside of the box and exploring options that have not been tried before. To address these weaknesses, it may be beneficial to incorporate a variety of problem-solving strategies, including both analytical and creative approaches. This may involve considering different perspectives, brainstorming new ideas, and testing potential solutions through trial and error. Additionally, it may be helpful to continuously evaluate the effectiveness of the proposed model and make adjustments as needed to improve its ability to solve case problems.

Based on Fig. 3, it appears that the proposed model has achieved the highest values on most of the datasets, representing better performance compared to the other comparative studies. For example, on the Click dataset, it achieved a value of 100%, signifying the best performance, followed by the Hybrid and FUD with approximately values of 95%. On the Detec dataset, the proposed model and Hybrid model have achieved the highest values of 100 %and 96%, respectively, demonstrating the best act, while the RCNN and FUD models have achieved lower values of 93% and 95%, respectively. However, on the Onlin dataset, the Hybrid model has achieved the maximum value of 97%, suggesting better result compared to the others, while our model has reached 9%8, which is slightly lower. On the Face dataset, the proposed model accomplished the thoroughgoing value of 99%, giving the best outcome, while the rest models accomplished lower values ranging from 93% to 95%. The proposed model has performed the unsurpassed upshot on most of the datasets, while the Hybrid and FUD representations have also made well on some datasets. However, the RCNN looks to have completed lower standards, telling relatively lower recital compared to the other reproductions on maximum of the datasets as shown in Fig. 4.

From Fig. 5, we can see that for the Bitcoin dataset, the proposed model and Hybrid model spent the lowest time of 0.00240 seconds, this shows the better consequence compared to RCNN and FUD. Likewise, for the Click dataset, the proposed model has gotten 0.00280 seconds that is lowest, followed by the Hybrid model (0.00420 secs). RCNN and FUD took longer time, this means that is lower performance compared to the other methods.

Table 1. Dataset.

	Dataset	Cases	Year
Bitcoin	Bitcoin Heist Ransomware Address Dataset	2,916,697	2020
Click	Clickstream data for online shopping	165,474	2019
Detec	Detection of IoT botnet attack	7,062,606	2018
Face	Facebook Live Sellers in Thailand	7,051	2019
Onlin	Online Retail II	1,067,371	2019
Query	Query Analytics Workloads Dataset	260,000	2019

Table 2. Irrelevant cases.

	Case Base	RCNN	FUD	Hybrid	Proposed Model
Bitcoin	2,916,697	2,654,194	2,741,695	2,800,029	2,829,196
Click	165,474	148,927	153,891	160,510	160,510
Detec	7,062,606	6,426,971	6,638,850	6,850,728	6,991,980
Face	7,051	6,487	6,628	6,839	6,980
Onlin	1,067,371	971,308	1,003,329	1,035,350	1,056,697
Query	260,000	239,200	247,000	252,200	252,200

Table 3. Relevant cases.

	Case Base	RCNN	FUD	Hybrid	Proposed Model
Bitcoin	2,916,697	262,503	175,002	116,668	87,501
Click	165,474	16,547	11,583	4,964	4,964
Detec	7,062,606	635,635	423,756	211,878	70,626
Face	7,051	564	423	212	71
Onlin	1,067,371	96,063	64,042	32,021	10,674
Query	260,000	20,800	13,000	7,800	7,800

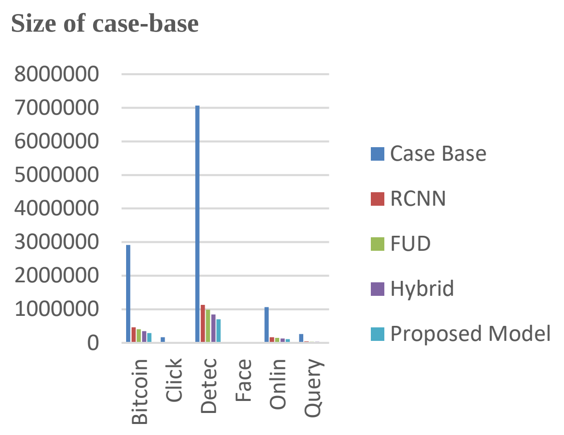

Fig. 2. Case-base size.

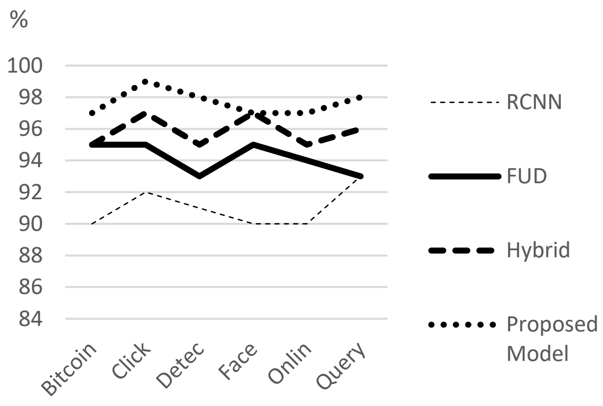

Fig. 3. Percent of Size Reduction.

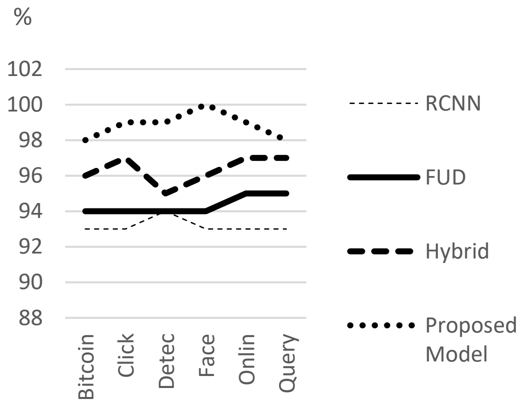

Fig. 4. Percent Problem-Solving.

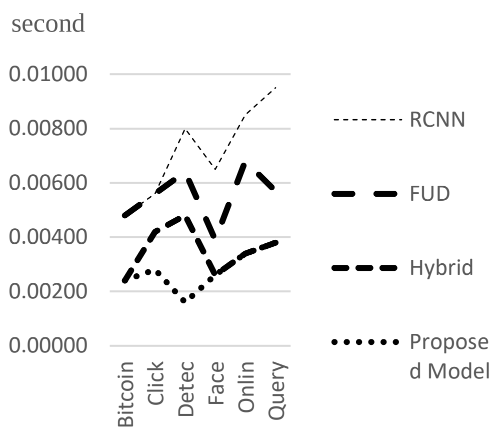

Fig. 5. Average problem-solving time.

For the Detec dataset, the proposed model used 0.00160 secs that the best performance is shown, followed by FUD with 0.00640 seconds. RCNN and Hybrid present the higher values demonstrating lower appearance. Similarly, for the Face dataset, the proposed model and FUD have achieved the lowest value of 0.00260, indicating better performance compared to RCNN and Hybrid. For the Onlin dataset, the Hybrid model has achieved the lowest value of 0.00340, followed by the proposed model with a value of 0.00340. RCNN and FUD have achieved higher values indicating lower performance. Finally, for the Query dataset, the proposed model has achieved the lowest value of 0.00380, indicating the best performance, followed by RCNN and FUD with values of 0.00950 and 0.00570, respectively. The Hybrid model has achieved a value of 0.00380, which is similar to the proposed model and indicates good performance. Consequently, the proposed model has performed well on most of the datasets, achieving the lowest value in several cases. The Hybrid model and FUD have also performed well in some cases. RCNN has generally achieved higher values indicating lower performance compared to the other models.

5. Conclusion

It seems that the proposed model is a promising solution for addressing the resource-intensive nature of case-based maintenance in case-based reasoning systems. By studying a combination of approaches such as the Relative Coverage Condensed Nearest Neighbour, Footprint utility deletion, and a Hybrid approach, the model is able to identify relevant and irrelevant cases, reduce the size of the case-base, and maintain the system's competency. The results of the proposed model are encouraging, as it is able to achieve a lower reduction size of the case-base compared to traditional studies while also providing a higher percentage of problem-solving success. Additionally, the average problem-solving time is shorter, which suggests that the model is more efficient in terms of computational resources.

Therefore, the proposed model appears to be a alternative for optimizing case-based reasoning systems. It may be worthwhile for researchers and practitioners to further explore and evaluate the effectiveness of this approach in various domains and applications.

References

[1] A. Lawanna, "Approval deletion model for case-based maintenance of case-based reasoning system," in *2018 IEEE 7th Global Conference on Consumer Electronics (GCCE)*, Nara, Japan, 2018, pp. 576-580, doi: 10.1109/GCCE.2018.8574871.

[2] A. Aamodt and E. Plaza, "Case-based reasoning: Foundational issues, methodological variations, and system approaches," *AI Communications*, vol. 7, no. 1, pp. 39-59, 1994.

[3] A. Lawanna and J. Wongwuttiwat, "Problem-based model for the improvement of case-based reasoning system," in *2017 14th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, Phuket, Thailand, 2017, pp. 349-352, doi: 10.1109/ECTICon.2017.8096245.

[4] L. Kangju, S. Weitang, L. Yefeng and Z. Yuan, "Research on self-maintenance strategy of CNC machine tools based on case-based reasoning," in *2021 3rd International Conference on Industrial Artificial Intelligence (IAI)*, Shenyang, China, 2021, pp. 1-5, doi: 10.1109/IAI53119.2021.9619222.

[5] J. Wang, Y. Xiang, and Y. Liu, "A case-based maintenance approach for power equipment based on deep learning," *IEEE Access*, vol. 7, pp. 129140-129149, 2019, doi: 10.1109/ACCESS.2019.2935624.

[6] R. Fornells, J. A. Rodríguez-Aguilar, and M. Esteva, "Solving multi-issue negotiation problems with case-based reasoning," *IEEE Transactions on Cybernetics*, vol. 50, no. 2, pp. 433-445, Feb. 2020, doi: 10.1109/TCYB.2018.2879224.

[7] M. Zhang, Y. Zhang, and H. Wang, "A case-based maintenance decision support system for bridge infrastructure," *IEEE Access*, vol. 9, pp. 4920-4930, 2021, doi: 10.1109/ACCESS.2020.3048679.

[8] M. J. Jeon, H. J. Jeong, and K. R. Lee, "A case-based maintenance decision support system for industrial facilities," *IEEE Access*, vol. 9, pp. 14309-14320, 2021, doi: 10.1109/ACCESS.2021.3059691.

[9] J. Han, J. Sun, W. Song and S. Li, "Prediction of surface roughness in machining operations using case-based reasoning and Euclidean distance," *IEEE Access*, vol. 7, pp. 149579-149590, 2019, doi: 10.1109/ACCESS.2019.2947483.

[10] T. V. N. Rao and D. R. Parhi, "Performance analysis of clustering based indexing in case-based reasoning for fault diagnosis of rotating machines," *IEEE Access*, vol. 8, pp. 19915-19924, 2020, doi: 10.1109/ACCESS.2020.2967689.

[11] H. Lee, C. Lee, and Y. Lee, "Development of a real-time predictive maintenance system for industrial equipment based on k-nearest neighbors," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 9, pp. 5962-5971, Sept. 2020, doi: 10.1109/TII.2020.3019863.

[12] C. Zeng, D. Li, and D. Chen, "A novel k-nearest neighbor-based predictive maintenance framework," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 5, pp. 3445-3454, May 2021, doi: 10.1109/TII.2020.3049333.

[13] J. Li, Z. Liu, X. Dong, and F. Zhao, "A case-based reasoning method for large-scale wind power equipment maintenance considering retrieval and adaptation time," *IEEE Access*, vol. 9, pp. 32756-32764, 2021, doi: 10.1109/ACCESS.2021.3067856.

[14] B. Smyth and M. T. Keane, "Remembering to forget," in *Proceedings of the 14th international joint conference on Artificial intelligence*, 1995, pp. 377-382.

[15] C. Zhang, H. Xu, L. Wang, and X. Lin, "A case-based reasoning approach for equipment maintenance based on improved coverage and reachability," *IEEE Access*, vol. 8, pp. 176047-176056, 2020, doi: 10.1109/ACCESS.2020.3021626.

[16] Y. Wu, Z. Zhang, and Q. Gao, "A case-based maintenance approach for injection molding machines based on coverage and reachability," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 4, pp. 2884-2894, April 2021, doi: 10.1109/TII.2020.3035679.

[17] S. Xiao, Y. Chen, and W. Li, "Fuzzy k-nearest neighbor algorithm-based case retrieval in aircraft maintenance," *IEEE Access*, vol. 7, pp. 81162-81171, 2019, doi: 10.1109/ACCESS.2019.2924721.

[18] Y. Xu, J. Li, and Y. Zhang, "An intelligent maintenance method of equipment based on fuzzy k-NN algorithm," *IEEE Access*, vol. 8, pp. 188191-188202, 2020, doi: 10.1109/ACCESS.2020.3030767.

[19] A. Singh and A. Kumar, "A dynamic memory management scheme for case-based reasoning," *IEEE Access*, vol. 9, pp. 11909-11922, 2021, doi: 10.1109/ACCESS.2021.3059015.

[20] Y. Zhou, G. Yang, J. Lu and X. Liu, "An improved condensed nearest neighbor algorithm for online case-based reasoning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 3, pp. 902-913, March 2020, doi: 10.1109/TNNLS.2019.2905765.

[21] X. Zhang and Y. Hu, "Case selection based on improved condensed nearest neighbor rule for case-based reasoning," *IEEE Access*, vol. 8, pp. 93149-93158, 2020, doi: 10.1109/ACCESS.2020.2992822.

[22] A. De Santi, D. Guiducci and L. Ippoliti, "Memory Management and Adaptation in Case-Based Reasoning for Condition-Based Maintenance," *in IEEE Transactions on Industrial Electronics*, vol. 67, no. 2, pp. 1536-1545, Feb. 2020, doi: 10.1109/TIE.2019.2914257.

[23] B. Smyth and E. McKenna, "Competence models and the maintenance problem," *Computational Intelligence*, vol. 17, no. 2, pp. 235-249, 2001.

[24] S. Hao, Y. Zhang, and H. Wang, "A novel hybrid case-based reasoning approach with similarity and

diversity evaluation for smart manufacturing," *Robotics and Computer-Integrated Manufacturing*, vol. 63, p. 101947, 2001.

[25] A. Garg, M. Gupta, and J. P. Singh, "An approach for dynamic case adaptation in case-based reasoning systems," *Journal of Intelligent & Fuzzy Systems*, vol. 36, no. 1, pp. 677-691, 2019.

[26] S. S. Madkour and M. A. El-Dosuky, "A case-based maintenance approach using relative coverage," in *2019 11th International Conference on Computer and Automation Engineering (ICCAE)*, Cairo, Egypt, 2019, pp. 31-36.

[27] S. S. Madkour and M. A. El-Dosuky, "Case-based maintenance for rotating machinery using relative coverage," *IEEE Transactions on Industrial Electronics*, vol. 66, no. 9, pp. 7077-7086, Sept. 2019.

[28] S. S. Madkour and M. A. El-Dosuky, "A case-based approach for fault diagnosis and prognosis of rotating machinery using relative coverage," *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1-10, 2021.

[29] D. B. Leake and D. C. Wilson, "Remembering why to remember: Performance-guided case-base maintenance," in *Advances in Case-Based Reasoning: 5th European Workshop, EWCBR 2000* Trento, Italy, September 6–9, 2000, pp. 161-172.

[30] H. Li, J. Li, and J. Zhou, "A dynamic case-based reasoning approach for equipment maintenance scheduling," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 48, no. 4, pp. 587-598, 2018.

[31] W. Gao, L. Zou, and Y. Zhang, "A dynamic case-based reasoning method for fault diagnosis of industrial equipment based on expert knowledge and data," *IEEE Access*, vol. 8, pp. 109259-109272, 2020.

[32] J. Kwan and T. Dao, "Streaming-based case selection for predictive maintenance," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, Portland, OR, USA, 2019, pp. 85-92.

[33] F. Tang, Q. Zhao, and Y. Xu, "A case retention policy for case-based reasoning in equipment diagnosis," in *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, Tianjin, China, 2019, pp. 1752-1757.

[34] M. Lu, J. Zhang, and X. Jiang, "A deletion policy for case-based reasoning in predictive maintenance," in *2019 IEEE International Conference on Mechatronics and Automation (ICMA)*, Tianjin, China, 2019, pp. 2167-2172.

[35] X. Huang, X. Xu, Z. Sun, and Y. Liu, "An effective random elimination technique for case-based reasoning in predictive maintenance," in *2021 IEEE International Conference on Industrial Technology (ICIT)*, Lyon, France, 2021, pp. 197-202.

[36] Z. Zheng, Y. Lu, and Y. Zhao, "An improved case addition and reuse approach for machine fault diagnosis based on case-based reasoning," *IEEE Access*, vol. 9, pp. 41983-41996, 2021.

Asst. Prof. Dr. Thacha Lawanna is currently a lecturer in International College of Digital Innovation at Chiang Mai University Thailand. She completed her Ph.D. degree in Information Technology from Assumption University, Thailand. Her expertise includes test case selection, software testing and data mining, with a number of scholarly works have been published in international reputable journals.

Asst. Prof. Dr. Rujira Ouncharoen as the Dean of the International College of Digital Innovation at Chiang Mai University in Thailand. She obtained her doctoral degree in Mathematics from Mahidol University in Thailand. Her areas of specialization include the SIR Model and Mathematics Model, and she has authored numerous scholarly publications in respected international journals.