TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Pilleriin Kõiva 193413IAIB

Implementation and Performance Assessment of Swarm Intelligence Based Numerical Association Rule Mining Algorithms

Bachelor's thesis

Supervisor: Minakshi Kaushik

M.Tech,
Early Stage Researcher

Tallinn 2022

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Pilleriin Kõiva 193413IAIB

Kollektiivsel intelligentsusel põhinevate numbriliste assotsiatsioonireeglite kaevandamise algoritmide realiseerimine ja soorituse hindamine

Bakalaureusetöö

Juhendaja: Minakshi Kaushik

M.Tech,
doktorant-nooremteadur

Tallinn 2022

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Pilleriin Kõiva

30.05.2022

Abstract

Association rule mining (ARM) is an important technique for discovering relationships among different data items in data mining. The classical ARM has limitations with numerical data as these algorithms can only deal with binary data. Numerical association rule mining (NARM) is an extended version of ARM that identifies association rules (ARs) in numerical data items. Many evolutionary and swarm intelligence-based (SI) algorithms are proposed for extracting significant rules from a numeric dataset without discretizing numeric attributes under the umbrella of optimization algorithms.

Several of these algorithms claim to be more efficient than others. Therefore, it is hard to determine the most efficient algorithm to mine numerical association rules among various NARM algorithms. This thesis examines the performance assessment of four swarm intelligence-based numerical association rule mining algorithms (MOPAR, MOCANAR, ACO_R and MOB-ARM). These NARM algorithms are proposed by researchers in papers, while their performance assessment in the real world is missing. However, these algorithms are compared with other generic optimization algorithms in different categories. In this thesis, author implements algorithms, and the implementations are validated. To the best of our knowledge, this is the first time the performance of multi-objective NARM optimization algorithms within the SI category has been assessed and compared. To find out which algorithm is the best, the algorithms are tested with four datasets, and parameters are determined for fair comparison under equal conditions. Additionally, the algorithms are validated with two datasets, which expose some differences in the results of the implementations and original algorithms. The performance assessment of the implementations shows that different multi-objective NARM optimization algorithms are suitable for different needs. The performance assessment shows that the main problems are the need for parameter modification in some datasets and that the algorithms are time-consuming.

This thesis is written in English and is 35 pages long, including 7 chapters, 15 figures and 13 tables.

Annotatsioon

Kollektiivsel intelligentsusel põhinevate numbriliste assotsiatsioonireeglite kaevandamise algoritmide realiseerimine ja soorituse hindamine

Assotsiatsioonireeglite kaevandamine on oluline andmekaeve meetod erinevate andmeüksuste vaheliste seoste avastamiseks. Klassikalisel assotsiatsioonireeglite kaevandamisel on arvandmetega seotud piirangud, kuna need algoritmid saavad käsitleda ainult binaaseid arvandmeid. Numbriliste assotsiatsioonireeglite kaevandamine on assotsiatsioonireeglite kaevandamise laiendatud versioon, mis tuvastab numbrilistes andmeüksustes assotsiatsioonireegleid. Mitmeid evolutsioonil ja kollektiivsel intelligentsusel põhinevaid algoritme on välja pakutud selleks, et arvulistest andmekogumitest eraldada olulisi reegleid ilma, et peaks numbrilisi atribuute diskretiseerima.

Mitmed neist algoritmidest väidavad end olevat teistest tõhusamad. Seetõttu on erinevate numbriliste assotsiatsioonireeglite kaevandamise algoritmide hulgast raske määrata kõige tõhusamat algoritmi numbriliste assotsiatsioonireeglite kaevandamiseks. Käesolevas lõputöös hinnatakse nelja kollektiivsel intelligentsusel põhineva numbriliste assotsiatsioonireeglite kaevandamise algoritmi sooritust (MOPAR, MOCANAR, ACO_R ja MOB-ARM). Need algoritmid on välja pakutud teadlaste poolt dokumentides, samas kui nende soorituse hindamine tegelikus maailmas puudub. Neid algoritme võrreldakse aga teiste üldiste optimeerimisalgoritmidega erinevates kategooriates. Selles lõputöös realiseerib autor algoritme ja algoritmid on valideeritud. Meie teadmiste kohaselt on see esimene kord kui kollektiivsel intelligentsusel põhinevaid mitme eesmärgiga numbriliste assotsiatsioonireeglite kaevandamise optimeerimisalgoritmide sooritusi hinnatakse ja võrreldakse. Parima algoritmi väljaselgitamiseks testitakse algoritme nelja andmekogumiga ja määratakse parameetrid õiglaseks võrdlemiseks võrdsetel tingimustel. Lisaks valideeritakse algoritme kahe andmekogumiga, mis paljastavad mõningad erinevused realiseerimiste ja algsete algoritmide tulemustes. Rakenduste soorituse hindamine näitab, et erinevatele vajadustele sobivad erinevad algoritmid.

Soorituse hindamine näitab, et peamised probleemid on parameetrite muutmise vajadus mõnes andmekogumis ja see, et algoritmid on aeganõudvad.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 35 leheküljel, 7 peatükki, 15 joonist, 13 tabelit.

List of abbreviations and terms

ARM	Association Rule Mining
NARM	Numerical Association Rule Mining
PSO	Particle Swarm Optimization
CS	Cuckoo Search
ACO	Ant Colony Optimization
BA	Bat Algorithm
MOPAR	Multi-objective particle swarm optimization algorithm for numerical association rule mining
MOCANAR	Multi-objective cuckoo search algorithm for numerical association rule mining
ACO_R	Multi-objective ant colony optimization algorithm for continuous domains
MOB-ARM	Multi-objective bat algorithm for numerical association rule mining
SI	Swarm Intelligence

Table of contents

List of figures

List of tables

1 Introduction

Nowadays, a lot of data is stored in transactional databases, where each column represents an attribute, and each row represents an instance. In data mining, Association Rule Mining (ARM) is a well-known technique to find interesting relations among various data items [1].

The ARM was introduced by Agrawal in 1993 [2] to discover the associations between data items in market basket analysis. Later, some essential algorithms, such as Apriori [3], and FP-Growth [4], were proposed. These algorithms were suitable for binary data but could not deal with numerical data. In 1994, Srikant introduced the concept of quantitative association rule mining (QARM) [5] to deal with numerical data. Further, this technique is also known as numerical association rule mining (NARM) [6].

Several methods such as optimization, discretization, and distribution are available in the literature to solve the problem of NARM [6], [7]. The optimization method seems to be a potential solution to deal with such complex problems. Evolutionary-based and swarm intelligence-based algorithms come under the optimization method [7].

Recent NARM optimization algorithms cover swarm intelligence-based algorithms, which are based on animal, and insect movements and the biological behaviour of natural objects [8].

Several SI-based NARM algorithms claim to be more efficient than other algorithms in the literature. However, these algorithms are compared with evolutionary algorithms, not SI-based ones. Primarily, these algorithms are proposed by researchers in papers theoretically, while their implementations and performance assessment in the real world are missing [8].

The problem of finding the best-performing SI-based NARM algorithm on real-world data motivates the implementation and performance analysis of these algorithms.

The goal of this work is to implement and analyse the performance of four SI-based algorithms, which are MOPAR [9], ACO_R [10], MOB-ARM [11], and MOCANAR [12].

This goal is divided into the following tasks: 1) implement the chosen algorithms in Python, 2) validate the implementations of the algorithms, and 3) assess the performance of these algorithms using the number of rules, time consumed, support, confidence, comprehensibility, and interestingness measures. As a result of this work, the chosen variants are implemented, and the performance of the implementations is compared and analysed. Based on the analysis, it can be concluded which algorithm is the best.

This thesis is organized as follows: In section 2, related work is discussed. In section 3, the theoretical background of data mining, association rule mining, numerical association rule mining, and swarm intelligence optimization is given. In section 4, swarm intelligence optimization algorithms, their detailed information of the chosen algorithm variants is given, and their implementations are explained. In section 5, the additional implementation necessary to run the algorithms and experimental setup are given. The validation and performance evaluation of algorithms are shown in section 6. Finally, a summary is presented in section 7.

2 Related work

Works related to the performance assessment of NARM algorithms are discussed in this section. These works differ from our work in the assessed algorithms and the measures that are used to assess their performances.

Performance analysis of evolutionary algorithms for NARM was done in [6]. This work analysed the performance of seven evolutionary and fuzzy evolutionary NARM algorithms. The chosen algorithms were also compared against the Apriori algorithm. A comparative analysis was done in terms of support, confidence, the number of rules mined, number of records covered, and time spent using eleven real-world datasets. This research found that the evolutionary algorithms have better results in terms of support, confidence, and time metrics. According to the authors, this was the first time performance analysis of evolutionary and fuzzy evolutionary algorithms for NARM was done with real-world datasets.

Altay performed a performance analysis of multi-objective NARM algorithms in [13]. In this research, six multi-objective and four single-objective optimization algorithms were chosen to be compared. The number of rules, coverage percentage, support, confidence, conviction, lift, netconf, ylesQ and certain factor measures were used for comparative analysis. Ten real-world datasets were used. This research found that multi-objective algorithms outperformed single-objective algorithms in terms of support, lift, certain factor, netconf, and yulesQ metrics. According to the authors, this was the first time performance analysis of multi-objective NARM optimization algorithms was done with real-world datasets.

An example of using NARM for real-world problems was presented in [14], which did an association analysis of multi-objective NARM algorithms using data about Parkinson's disease. This research used numerical data consisting of speech samples related to Parkinson's disease. This data was used on three multi-objective NARM algorithms to find association rules related to healthy individuals and patients with Parkinson's disease. The number of rules, coverage percentage, support, confidence,

conviction, lift, netconfylesQ and certain factor measures were used for comparative analysis. According to the authors, this was the first time an association analysis of Parkinson's disease for early diagnoses was performed.

Another example of using NARM for real-world problems was presented in [15], which presented an association analysis of multi-objective NARM algorithms using data about liver fibrosis. This data was used on two multi-objective NARM algorithms to find association rules related to liver fibrosis. The number of rules, coverage percentage, support, confidence, conviction, lift, netconf, ylesQ and certain factor measures were used for comparative analysis. After that, a sensitivity analysis was done to find the best parameters for this problem. According to the authors, this was the first time an association analysis of liver fibrosis for early diagnosis was performed.

3 Background

In this section, the theoretical background of numerical association rule mining is explained to aid comprehension of the implementation and performance analysis of the algorithms. First, an overview of data mining and association rule mining is given. After that, numerical association rule mining and swarm intelligence optimization are explained. Finally, multi-objective NARM is discussed.

3.1 Data mining

Data mining is the process of extracting previously unknown and potentially useful information from data in databases. There are many different challenges that data mining techniques deal with, such as handling different types of data, and the efficiency and scalability of data mining techniques [16].

Data mining is application-dependent, and different applications require different mining techniques. Examples of different applications include association rule mining, data generalization, data classification, and data clustering [16].

3.2 Association Rule Mining

Association rule mining is a data mining technique that aims to extract interesting correlations, frequent patterns, or associations among sets of items in mainly transactional databases [1]. Association rule mining was first introduced by [2]. In ARM, association rules are if-then relationships, that have an antecedent, and a consequent [1].

One application of association rule mining is to find out what products are bought together from a store. The mined association rules can help determine how to boost the sales of a product, what products may be impacted by discontinuing another product, and the best locations for the products [2].

A commonly used algorithm for ARM is Apriori [3]. The Apriori algorithm works by constructing a candidate set of frequent itemsets, counting occurrences of each candidate

itemset and determining frequent itemsets based on pre-determined support and confidence [16]. Other known ARM algorithms are AIS [2] and FP-growth [4].

3.3 Numerical Association Rule Mining

Numerical association rule mining is used to extract association rules from numerical data. An example of a numerical association rule $\text{Age} \in [25, 40] \Rightarrow \text{Salary} \in [1300, 2000]$ would be "If the age of an employee is between 25 and 40, then their salary is between 1300 and 2000." Here, age is the antecedent and salary is the consequent part of the rule. Numerical association rule mining can be divided into distribution, discretization, and optimization methods [7].

3.3.1 Swarm Intelligence Optimization

Optimization methods provide a robust and effective approach for massive search spaces. This method is divided into biology-inspired and physics-based methods. Biology-based algorithms are further divided into swarm intelligence- and evolution-based algorithms [7].

Swarm intelligence optimization techniques are inspired by the collective behaviour of social insects, birds, or animals, that follow a set of rules [17]. According to [8], the most popular swarm intelligence-based algorithms for NARM are Particle Swarm Optimization and Ant Colony Optimization. The Bat Algorithm and the Cuckoo Search Algorithm are also part of the family.

The pseudocode of a nature-inspired metaheuristic algorithm is shown in Figure 1. First, a population of agents is initialized with random solutions. The solutions are evaluated in terms of the used objectives. After that, each agent modifies its solution until a stopping criterion is met, and the best generated solutions are returned [8].

```
Initialize population
Evaluate solutions
For iteration in max iterations:
      Modify solutions
      Evaluate modified solutions
      Select best solutions
Return best solutions
```

Figure 1. Pseudocode of nature-inspired metaheuristic algorithm.

A solution needs to be encoded in the search space to mine numerical association rules. When using the Michigan approach for representing individuals, each individual encodes a single association rule [8].

The first representation of association rules used in NARM, shown in equation (1), encodes the rule as a vector of attributes consisting of n number of triplets, where n is the number of attributes in the transactional database. Each triplet consists of three elements. ACN_n determines whether the attribute is present in the rule. ACN stands for antecedent, consequent, not present. LB_n determines the lower bound of the attribute and UB_n determines the upper bound of the attribute [8].

$$((ACN_1, LB_1, UB_1), \dots, (ACN_n, LB_n, UB_n)) \quad (1)$$

Another way to represent a rule as a vector is shown in equation (2). Here, s_n shows the value and δ_n shows the standard deviation of the attribute [10].

$$((ACN_1, s_1, \delta_1), \dots, (ACN_n, s_n, \delta_n)) \quad (2)$$

The ACN_n element can be encoded in two different ways. In the first way, shown in equation (3), if ACN_n value is less than or equal to 1/3, then the attribute is in the antecedent part of the rule. If the value is bigger than 1/3 and smaller than or equal to 2/3, then the attribute is in the consequent part. If the value is bigger than 2/3, then the attribute is not present in the rule [8].

$$j = \begin{cases} ACNj \leq \frac{1}{3}, & antecedent \\ \frac{1}{3} < ACNj \leq \frac{2}{3}, & consequent \\ ACNj > \frac{2}{3}, & not\ present \end{cases} \quad (3)$$

The second way to encode ACN_n is shown in equation (4). Here, if ACN_n is 1, then the attribute is in the antecedent part, if it is 2, then it is in the consequent part, and if it is 0, then the attribute is not present in the rule [12].

$$j = \begin{cases} ACNj = 1, & antecedent \\ ACNj = 2, & consequent \\ ACNj = 0, & not\ present \end{cases} \quad (4)$$

Another way to represent an association rule is shown in equation (5), where *n* is the number of attributes in the database. Here, cp_i defines the cutting point between the antecedent and consequent attributes. If the $o_{i,n}$ value is zero, then the attribute is omitted

from the rule, otherwise it represents the id of the interval of the attribute [8]. In this case, the database is discretized into intervals.

$$(cp_i, o_{i,1}, \dots, o_{i,n}) \tag{5}$$

Fitness functions are an important part of evaluating mined association rules to estimate the quality of solutions [8]. The measures used in this thesis are support, confidence, comprehensibility, and interestingness.

The support of an association rule $X \Rightarrow Y$ determines how frequently the itemset appears in a transactional database. The equation is shown in equation (6), where $|X \cup Y|$ designates the number of transactions containing the antecedent X and consequent Y part of the rule, and $|D|$ designates the number of transactions in the database [8].

$$supp(X \Rightarrow Y) = \frac{|X \cup Y|}{|D|} \tag{6}$$

The confidence of an association rule $X \Rightarrow Y$, shown in equation (7), shows how many transactions that contain X, also contain Y [8].

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)} \tag{7}$$

According to [18], if the number of conditions involved in the antecedent part is less than the consequent part, the rule is more comprehensible. To calculate the comprehensibility of an association rule $X \Rightarrow Y$, equation (8) is used. Here $|Y|$ represents the number of attributes in the consequent part of the rule and $|X \cup Y|$ shows the number of attributes in both the antecedent and consequent parts of the rule [18].

$$comp(X \Rightarrow Y) = \frac{\log(1+|Y|)}{\log(1+|X \cup Y|)} \tag{8}$$

The interestingness measure is focused on discovering hidden information by extracting interesting rules. The equation (9) consists of three parts. The first part shows the probability of generating the rule based on the antecedent part. The second part shows the probability based on the consequent part. The third part shows the probability of not generating the rule based on the whole dataset [18].

$$inter(X \Rightarrow Y) = \frac{supp(X \Rightarrow Y)}{supp(Y)} \cdot \frac{supp(X \Rightarrow Y)}{supp(X)} \cdot \left(1 - \frac{supp(X \Rightarrow Y)}{|D|}\right) \tag{9}$$

3.4 Multi-objective NARM

When an optimization problem involves only one objective function, it is called single-objective. It is called multi-objective when it involves more than one objective functions [19].

The weighted sum method is a classical multi-objective method. This method summarizes multiple objectives into a single objective by multiplying each objective with a pre-defined weight. It is the simplest multi-objective method, but finding the right multipliers can be challenging [19].

Another method is Pareto dominance, in which all the objectives are evaluated simultaneously. One solution dominates another if it improves one objective without causing a worse outcome for all the other objectives. Using this dominance criterion, non-dominated solutions can be defined [19].

4 Swarm Intelligence Optimization Algorithms

This section explains the chosen swarm intelligence optimization methods and algorithms. The chosen methods are Particle Swarm Optimization, Cuckoo Search, Bat Algorithm, and Ant Colony Optimization.

4.1 Particle Swarm Optimization

In this section, Particle Swarm Optimization (PSO) is explained. First, the theoretical background of PSO is given in 4.1.1. After that, the theory of the chosen variant, MOPAR, is described in 4.1.2. Finally, the implementation of MOPAR algorithm is given in 4.1.3.

4.1.1 PSO background

Particle Swarm Optimization (PSO) is inspired by the behaviour of social animals, such as flocks of birds. A flock of birds follow a set of rules, which are used in PSO algorithms. For example, birds try to avoid collisions, move closely together, and share information with each other. The agents in PSO are particles, that have been assigned a set of positions and velocities, based on which the particles move during each iteration [20].

According to [8], many different PSO algorithms have been proposed for NARM. Some examples are PARCD [21], which uses Cauchy distribution, MOPAR [9], which uses rough values for NARM, and MOPSO [22], which uses discretization to deal with numerical attributes.

4.1.2 MOPAR

A multi-objective PSO algorithm for NARM was proposed in [9]. MOPAR uses confidence, comprehensibility, and interestingness objectives to evaluate association rules. For rule encoding, equations (1) and (3) are used. Pareto optimality is used for extracting non-dominated rules [9].

The pseudocode for MOPAR, which is based on [9], is shown in Figure 2. First, the population, which consists of particles, the external repository, which consists of the

mined rules, and the global best, which is the best particle, are initialized. In each iteration, the particle population is updated. After that, the best solutions from the population are added to the external repository, and the global best solution is updated. Finally, after the iterations, the external repository is returned [9].

```
Initialize population, external repository, global best
For iteration in maximum iterations:
      Update particles in population
      Update external repository, global best
Return external repository
```

Figure 2. Pseudocode of MOPAR.

To update particles, equations (10) and (11) are used, which update the velocities and positions of a particle. After that, the particle's objectives are evaluated. Finally, the local best solution of each particle is updated using Pareto dominance [9].

$$v_{i,k}(t+1) = w(t)v_{i,k}(t) + c_1R_1\left(lbest_{i,k}(t) - x_{i,k}(t)\right) + c_2R_2\left(gbest_{i,k}(t) - x_{i,k}(t)\right) \quad (10)$$

$$x_{i,k}(t+1) = x_{i,k}(t) + v_{i,k}(t+1) \quad (11)$$

To find a global best solution, roulette wheel selection is used. The roulette wheel first assigns a rank to each particle using equation (12), in which xRank is a user-specified parameter and local dominated count is the number of a particle's local best solutions that the current solution dominates. After that, each particle is assigned a probability based on equation (13). Based on these probabilities, a particle is chosen [9].

$$rank_i(t) = \frac{xRank}{local\ dominated\ count} \quad (12)$$

$$Prob_i(t) = \frac{rank_i(t)}{\sum_{k=1}^{n} rank_k(t)} \quad (13)$$

4.1.3 MOPAR algorithm

For the implementation of MOPAR, two classes are necessary. These classes are ParticleSearchOptimization and Particle. ParticleSearchOptimization has the methods necessary to run the algorithm, and that are unique to the algorithm. The Particle class is the representation of the agent that MOPAR uses, and holds the methods used for modifying the particles. In the ParticleSearchOptimization class, the main method is the

multi_objective_particle_search_optimization_algorithm, which generates rules based on the MOPAR algorithm. Figure 3 shows the steps of the algorithm and is based on [9].

Input	Data, population size, maximum iterations, external repository size, c1, c2, inertia weight, xRank
Output	External repository
Step 1	Initialize population. Evaluate the objectives of the generated rules. Initialize external repository. Initialize global best.
Step 2	Update the velocities of particles. Update the positions of particles and evaluate the objectives of the new rules. Update the non-dominated local set of each particle. Update local best.
Step 3	Update external repository. If the size of the external repository is bigger than external repository size, then particles that dominate more rules are removed.
Step 4	Update global best by using the roulette wheel selection.
Step 5	If maximum number of iterations is not reached, go to step 2
Step 6	Return the external repository

Figure 3. MOPAR algorithm steps.

To run the MOPAR algorithm, the class ParticleSearchOptimization is initialized with the parameters shown in the input row of Figure 3. Data is the data frame used to mine rules. Population size is the number of particles that are generated and modified with each iteration. Maximum iterations parameter is the number of times the particle population is modified. External repository size is the maximum number of rules that are mined. Parameters c1, c2 and inertia weight are used as multipliers to calculate new velocities for the particles. Velocity limit is the maximum value for the particle velocity. Parameter xRank is used for calculating the particle rank in the population.

First, step 1 of the algorithm is executed, in which the population is initialized with population size random particles. For the external repository, non-dominated rules from the population are found, and the global best solution is initialized using the roulette wheel selection.

In each iteration, steps 2-4 are executed. In step 2, the particle population is updated, by calculating new velocities and positions of each particle. If the new rule dominates the local best solution, then the new rule is added to the non-dominated local set and the rule is set as the local best solution. In step 3, the external repository is updated by merging the current external repository with the population and finding the non-dominated rules

from this merged set. If the external repository exceeds its maximum size, particles that dominate more rules are removed. In step 4, the global best is updated using the roulette wheel selection if the external repository is not empty; otherwise, a particle with the highest fitness is used. In step 6, after the iterations are done, the external repository is returned, which consists of all the best rules that were generated.

The additional methods in ParticleSearchOptimization are used to initialize the population and generate rules and velocities. The implementation includes methods for updating particles, external repository, and global best, as well as the implementation of roulette wheel selection. Additionally, methods that check and fix a rule and evaluate objectives are implemented.

Each rule generated by the MOPAR algorithm is kept in a class called Particle. This class is used to store data about the generated rule. This class has multiple parameters. Rule is the mined association rule. A list of velocities defines the movement of the particle positions. Support, confidence, comprehensibility, and interestingness parameters hold the fitness values of the rule. Local best is the best rule that has been generated by the particle. Local non-dominated consists of all the local best solutions this rule has generated. Rank is used for calculating the probability of a particle. Probability is used in the roulette wheel selection. The dominating counter is the number of particles the current rule dominates.

The methods in the Particle class are used to update the particle in each iteration of the algorithm. These methods calculate the velocities and positions of the rule, determine whether a new rule is the local best rule, and calculate how many rules the current rule dominates in the local non-dominated list.

4.2 Cuckoo Search

Cuckoo Search (CS) is explained in this section. First, the theoretical background of CS is given in 4.2.1. After that, the theory of the chosen variant, MOCANAR, is explained in 4.2.2. Finally, the implementation of MOCANAR algorithm is given in 4.2.3.

4.2.1 CS background

Cuckoo Search (CS) is based on the lives of cuckoos. The cuckoo is a famous brood parasite bird. Brood parasites lay eggs in the nests of other birds, so the nest owner takes care of the brood parasite's eggs. This phenomenon is adapted into CS algorithms. The agents in CS are cuckoos [12].

There are not many applications of CS for NARM. One example is MOCANAR [12], which claims to be the first multi-objective and the first ARM application of CS.

4.2.2 MOCANAR

A multi-objective cuckoo search algorithm, MOCANAR, that extracts rules from numeric datasets was proposed in [12]. The objectives considered for MOCANAR are support, confidence, interestingness, and comprehensibility. For rule encoding, equations (1) and (4) are used. Pareto optimality is used for extracting non-dominated rules [12].

The pseudocode of MOCANAR, which is based on [12], is shown in Figure 4. In each increment, the population, which consists of cuckoos, and current non-dominated rules are initialized. In each generation, random cuckoos are generated and directed towards the best solution, using levy flight. After that, each cuckoo is directed towards the best solution, using levy flight. At the end of each generation current non-dominated rules are updated. At the end of each increment, final non-dominated rules are updated. Finally, the final non-dominated rules are returned [12].

```
Initialize final non-dominated rules
For increment in number of increments:
      Initialize population
      Initialize current non-dominated rules
      For generation in maximum generations:
            Generate and direct random cuckoos
            Generate eggs by directing cuckoos
            Generate new population
            Update current non-dominated rules
      Update final non-dominated rules
Return final non-dominated rules
```

Figure 4. Pseudocode of MOCANAR.

A tournament is used to choose the best solution when generating eggs. For this, a number of tournament cuckoos are selected randomly from the population, and a random non-dominated solution from this selection is returned [12].

A levy flight policy is used to direct cuckoos towards the best cuckoo. For each attribute of a source cuckoo's rule, three-step sizes are calculated using levy distribution and a target cuckoo. Based on these step sizes, the rule of a source cuckoo is modified [12].

To generate a new population, first, a percentage of eggs that have the worst support measure is eliminated. After that, the eggs and cuckoo population are merged into a temporary population. The temporary population is sorted in terms of support measure, and ¼ of the highest-ranking solutions are added to the new population. The same is done for the rest of the measures, after which a new population has been formed [12].

4.2.3 MOCANAR algorithm

For the implementation of MOCANAR, two classes are necessary. These classes are CuckooSearchOptimization and Cuckoo. CuckooSearchOptimization has the methods necessary to run the algorithm. The Cuckoo class is the representation of the agent that MOCANAR uses. The multi_objective_cuckoo_search_algorithm is the main method in the CuckooSearchOptimization class. Figure 5 shows the steps of the algorithm and is based on [12].

Input	Data, population size, number of increments, maximum generations, pa, pmut, number of tournament, number of random cuckoos, w1, w2, w3
Output	Final non-dominated rules
Step 1	Initialize population and cuckoo eggs. Evaluate the objectives of generated rules.
Step 2	Generate random cuckoos and direct them towards the best cuckoo in the population. Replace the worst cuckoo in the population with the directed cuckoo.
Step 3	Generate cuckoo eggs by directing all cuckoos in the population towards the best cuckoo. The best cuckoo is chosen with tournament.
Step 4	A percentage of worst eggs in terms of the support measure are eliminated. A new population is formed by choosing the cuckoos with the best objective measures.
Step 5	Population and non-dominated list are merged, and duplicated rules are deleted. Non-dominated rules from the merged list are assigned to the non-dominated list.
Step 6	If maximum number of generations is not reached, go to step 2
Step 7	Rules from non-dominated list are added to final non-dominated list.
Step 8	If maximum number of increments is not reached, go to step 1
Step 9	Duplicated and dominated rules are removed from final non-dominated.
Step 10	Return final non-dominated list

Figure 5. MOCANAR algorithm steps.

To run the MOCANAR algorithm, the class CuckooSearchOptimization is initialized with the parameters shown in the input row of Figure 5. Data is the data frame that is used for mining rules. Population size is the number of cuckoos that are randomly generated and modified each generation. The number of increments parameter is the number of times the population is initialized, and the generations are run. The maximum generation parameter is the number of times the random cuckoos and cuckoo eggs are generated and modified. Pa is the percentage of cuckoo eggs that are eliminated each generation. Pmut is the probability of mutation after the eggs are generated. The number of tournament is the number of cuckoos that are randomly selected in the tournament section. The number of random cuckoos is the number of cuckoos that are generated each generation. Parameters w1, w2, w3 are used in to specify the length of steps used for directing a cuckoo towards the best cuckoo.

Steps 1-7 of the algorithm are repeated number of increments times. First step 1 is executed, in which the population is initialized with population size number of random cuckoos. The cuckoo eggs parameter is initialized with an empty list.

Steps 2-5 are repeated maximum generations times. In step 2, random cuckoos are generated and directed towards the best cuckoo in the population. This is done by using the levy flight method. The directed cuckoo replaces the worst cuckoo in the population. In step 3, every cuckoo generates a cuckoo egg. For this, the best cuckoo is found by the tournament. After that, the current cuckoo is directed towards the chosen best cuckoo using the levy flight method. The new directed cuckoo is the generated cuckoo egg. Step 4 generates a new population. First, a percentage of cuckoo eggs is eliminated. Then, a new population is formed by choosing the best cuckoos in terms of the objectives. In step 5, the population and non-dominated lists are merged, and duplicated rules are removed. The non-dominated rules from this merged list are assigned to the current non-dominated list.

In step 7, the current non-dominated and final non-dominated rules lists are merged. A new distribution is generated for generating cuckoos. In step 9, the duplicated and dominated rules from the final non-dominated list are removed and the list is returned.

The additional methods in CuckooSearchOptimization are used to initialize the population and generate a random cuckoo. The implementation includes methods for

getting a new cuckoo by levy flights, mutating a cuckoo, getting best cuckoo with tournament, and choosing a new population. Additionally, methods that check and fix a rule and evaluate objectives are implemented.

Each rule generated by the MOCANAR implementation is kept in a class called Cuckoo. This class is used to store data about the generated rules. This class has multiple parameters. Rule is the mined association rule. Support, confidence, comprehensibility, and interestingness parameters hold the fitness values of the rule.

4.3 Ant Colony Optimization

In this section, Ant Colony Optimization (ACO) is explained. First, the theoretical background of ACO is given in 4.3.1. After that, the theory of the chosen variant, ACO_R, is explained in 4.3.2. Finally, the implementation of the ACO_R algorithm is given in 4.3.3.

4.3.1 ACO background

Ant Colony Optimization (ACO) is based on the behaviour of ant colonies. In many ant species, ants deposit a substance called pheromone on the ground, while walking to and from food. Other ants make decisions based on how strong the pheromone is on a given path. ACO uses this behaviour of ants to optimize solutions. The agents in the ACO are ants [23].

According to [8], ACO is one of the first members of the swarm intelligence-based family, besides PSO. An example is ACO_R [10], which was proposed as a multi-objective variant for NARM.

4.3.2 ACO_R

ACO_R is a multi-objective ACO algorithm for continuous domains, which was proposed in [10]. The objectives considered for ACO_R are support, confidence, interestingness, and interval. For rule encoding, equations (2) and (4) are used. A weighted sum is used to determine the best solutions [10].

The pseudocode of ACO_R, which is based on [10], is shown in Figure 6. First the archive, which consists of solutions, is initialized, and solutions are ranked. In each iteration, the weights and probabilities of solutions are calculated. Each ant chooses a solution based

on the assigned probabilities and generates a new solution by sampling a gaussian function. At the end of each iteration, the solutions in the archive are ranked and the worst solutions are removed. After the iterations, the archive is returned [10].

```
Initialize archive, rank solutions
For iteration in maximum iterations:
      Calculate weights of solutions
Calculate probabilities of solutions
Each ant chooses a solution,
Each ant generates a new solution
      Rank solutions, trim archive
Return archive
```

Figure 6. Pseudocode of ACO_R.

The interval objective, shown in equation (14), favours rules that have smaller intervals. Here, n is the number of attributes, max bound and min bound are the maximum and minimum values for the attribute in the database. UB_i and LB_i are the upper and lower bounds of an attribute in the rule [10].

$$int = \sum_{i=0}^{n} \frac{(UB_i - LB_i)}{\max bound_i - \min bound_i} \tag{14}$$

All the mentioned objectives are put together into a single objective function, shown in equation (15). Here α_1, α_2, α_3, and α_4 are input parameters of the algorithm [10].

$$objective = \alpha_1 \cdot supp + \alpha_2 \cdot conf - \alpha_3 \cdot inter - \alpha_4 \cdot int \tag{15}$$

To calculate the weights of solutions, equation (16) is used, where k is the number of solutions in the archive, j is the rank of the solution, and q is a user-specified parameter [10]. If q is small, best-ranked solutions are more preferred, and if it is large, the probability is more uniform [23].

$$\omega_j = \frac{1}{qk\sqrt{2\Pi}} e^{\frac{-(j-1)^2}{2q^2k^2}} \tag{16}$$

To calculate the probabilities of solutions, equation (17) is used, where the weight of a solution is divided by the sum of all the weights of the solutions [10].

$$p_j = \frac{\omega_j}{\sum_{r=1}^{k} \omega_r} \tag{17}$$

After an ant chooses a solution based on the probabilities, a new solution is sampled using equation (18). Here, δ is calculated using equation (19) and μ is the value of the chosen solution [10].

$$P(x) = g(x, \mu, \delta) = \frac{1}{\delta\sqrt{2\Pi}} e^{\frac{-(x-\mu)^2}{2\delta^2}} \tag{18}$$

In equation (19), Ɛ is a user-specified parameter, the higher it is, the lower the convergence speed of the algorithm [23]. Parameter k is the number of solutions in the archive, s_j^i is the value of the chosen solution [10].

$$\delta = \xi \sum_{r=1}^{k} \frac{\left|s_r^i - s_j^i\right|}{k-1} \tag{19}$$

4.3.3 ACO_R algorithm

For the implementation of ACO_R, two classes are necessary. These classes are AntColonyOptimization and Solution. AntColonyOptimization has the methods necessary to run the algorithm. The Solution class holds the data of a generated rule. In the AntColonyOptimization class, ant_colony_optimization_for_continuous_domains is the main method. Figure 7 shows the steps of the algorithm and is based on [10].

Input	Data, archive size, ant colony size, maximum iterations, alpha1, alpha2, alpha3, alpha4, alpha5, q, e
Output	Archive
Step 1	Initialize and sort the archive.
Step 2	Initialize weights, probabilities, and ants.
Step 3	Ant chooses a solution, and generates a new solution by sampling a gaussian function. The objectives of the new solution are evaluated.
Step 4	If number of ants that have generated a new rule is not reached, go to step 3.
Step 5	Add ant generated solutions to archive, sort and cut off
Step 6	If number of iterations is not reached, go to step 2.
Step 7	Delete duplicated rules from archive
Step 8	Return non-dominated rules from archive

Figure 7. ACO_R algorithm steps.

To run the ACO_R algorithm, the class AntColonyOptimization is initialized with the parameters shown in the input row of Figure 7. Data is the data frame that is used for mining rules. The maximum iterations parameter is the number of times that each ant generates a new rule. Archive size is the number of solutions that are generated randomly at the beginning, and the number of best rules that are kept for each iteration. Ant colony size is the number of ants that generate a new rule each iteration. Alpha1, alpha2, alpha3 and alpha4 are multipliers for different objective functions, which are used for calculating the objective measure of a rule. Alpha5 is the multiplier used for calculating the interval of the attribute value of a rule. Q is used to calculate the weight of a solution. E is used to calculate the standard deviation of an attribute in a rule.

In step 1 of the algorithm, the archive is initialized with the archive size number of random solutions. Then the archive is sorted by the objective measures of the solutions.

Steps 2-5 are repeated maximum iterations times. Step 2 initializes the weights and probabilities of the solutions. After that, the ants are initialized with an empty list.

In steps 3-4, each ant chooses a solution from the archive, using the probabilities calculated in the last step. New solutions are generated based on the chosen solutions, by sampling gaussian functions. Finally, the objectives of the new rules are evaluated.

In step 5, the solutions generated by the ants are added to the archive. The archive is sorted based on the objective measure, and the ant colony size number of worst solutions are removed. In step 7, duplicated rules is removed by the algorithm, and non-dominated rules are returned.

The additional methods in AntColonyOptimization are used to initialize the archive, generate a solution, initialize weights, and calculate probabilities. The implementation includes methods for sampling a gaussian function and calculating a new standard deviation. Additionally, methods that evaluate objectives and calculate the bounds of an attribute value are implemented.

Each rule generated by the ACO_R implementation is kept in a class called Solution. This class is used to store data about the generated rule. This class has multiple parameters. Rule is the mined association rule. Support, confidence, comprehensibility, and interestingness parameters hold the fitness values of the rule. Int objective parameter

holds the interval objective measure value. Objective holds the weighted sum objective measure value.

4.4 Bat Algorithm

In this section, the Bat Algorithm (BA) is explained. First, the theoretical background of BA is given in 4.4.1. After that, the theory of the chosen variant, MOB-ARM, is explained in 4.4.2. Finally, the implementation of the MOB-ARM algorithm is given in 4.4.3.

4.4.1 BA background

The Bat Algorithm (BA) is based on the echolocation behaviour of bats. Bats use echolocation to sense distance and know the difference between food and prey. Bats fly randomly with a set velocity at a position with a frequency, wavelength, and loudness. This behaviour is used in BA for the movement of bats, which are the agents [24].

According to [8], BA has rarely been applied to ARM problems. One application of BA for NARM is MOB-ARM [11], which uses discretization to deal with numerical attributes.

4.4.2 MOB-ARM

MOB-ARM is a multi-objective BA for ARM, which was proposed in [11]. The objectives considered for MOB-ARM are support, confidence, comprehensibility, and interestingness. For rule encoding, equation (5) is used. Prior to using the algorithm, data is discretized into intervals. The weighted sum is used to determine the best solutions [11].

The pseudocode of MOB-ARM, which is based on [11], is shown in Figure 8. First the population, which consists of bats, is initialized. In each iteration, objective weights are generated, and each bat's frequency, velocity and rules are updated. At the end of each iteration, the bats are ranked, and a new global best solution is chosen. After each iteration, each bat's best solution is recorded as a non-dominated solution. Finally, the non-dominated solutions are returned [11].

```
Initialize population
For point in pareto points:
      For iteration in maximum iterations:
            Generate weights
            Update bat frequency, velocity, rule
            Rank bats, update global best solution
      Record best solutions as non-dominated solutions
Return non-dominated solutions
```

Figure 8. Pseudocode of MOB-ARM.

To generate weights, equation (20) is used. Here, k is the number of objectives used, which in MOB-ARM is two. The weights are used for calculating an objective measure shown in equation (21), which uses two objectives [11].

$$\sum_{k=1}^{k} w_k = 1 \tag{20}$$

$$Obj(R) = w_1 Obj_1(R) + w_2 Obj_2(R) \tag{21}$$

The objective measure uses two separate objectives, which are calculated using equations (22) and (23). The equations use user-specified parameters α, β, γ and δ as weights for the support, confidence, comprehensibility, and interestingness measures [11].

$$Obj_1(R) = \alpha conf(R) + \frac{\beta supp(R)}{\alpha} + \beta \tag{22}$$

$$Obj_2(R) = \gamma comp(R) + \frac{\delta inter(R)}{\gamma} + \delta \tag{23}$$

To update a bat's frequency and velocity, equations (24) and (25) are used. First, the new frequency is calculated using a maximum velocity, which is the number of attributes in the dataset. After that, a new velocity is calculated using the maximum velocity, new frequency, and the previous velocity [11].

$$f_i^t = 1 + (f_{max})\beta \tag{24}$$

$$v_i^t = f_{max} - f_i^t - v_i^{t-1} \tag{25}$$

A new rule is generated using an algorithm proposed in [25]. The rules are generated based on the velocity, frequency and loudness of the bat. Velocity determines the starting position of the change in the rule, and frequency determines how many attributes are changed. If the loudness of the bat is less than a random number, the attribute value at index velocity is increased; otherwise it is decreased. If the value goes out of bounds, it is set to zero [25]. After the rule is generated, one item in the rule is changed if a random number is bigger than the rate [11].

If the new rule's objective is better than the old objective, the rule is accepted, and the loudness and rate of the bat are updated. Loudness is decreased by using equation (26). The rate is increased using equation (27), where r_i^0 is the initial rate of the bat and *t* is the current iteration [11].

$$A_i^{t+1} = \alpha A_i^t \tag{26}$$

$$r_i^{t+1} = r_i^0[1 - exp(-\gamma t)] \tag{27}$$

4.4.3 MOB-ARM algorithm

For the implementation of MOB-ARM, two classes are necessary. These classes are BatAlgorithm and Bat. BatAlgorithm has the methods necessary to run the algorithm. The Bat class is the representation of the agent that MOB-ARM uses. In the BatAlgorithm class, multi_objective_bat_algorithm is the main method. Figure 9 shows the steps of the algorithm and is based on [11].

Input	Data, population size, iterations, pareto points, alpha, beta, gamma, delta, minimum support, minimum confidence
Output	Non-dominated solutions
Step 1	Initialize population. Sort population. Initialize global best. Initialize non-dominated solutions list.
Step 2	Initialize weights.
Step 3	Update every bat's frequency, velocity, and generate a new rule.
Step 4	If random number is bigger than bat's rate, change one attribute in the new rule.
Step 5	Check and fix rule. Evaluate fitness.
Step 6	If new objective is bigger than old objective, then accept new rule, increase rate, and decrease loudness of bat.
Step 7	Sort population. Update global best.
Step 8	If number of iterations is not reached, go to step 3.
Step 9	Add best solutions to non-dominated solutions
Step 10	If number of pareto points is not reached, go to step 2.
Step 11	Return non-duplicated rules from non-dominated solutions list

Figure 9. MOB-ARM algorithm steps.

To run the MOB-ARM algorithm, the class BatAlgorithm is initialized with the parameters shown in the input row of Figure 9. Data is the data frame that is used for mining rules. Population size is the number of random bats that are generated and

modified. Iterations are the number of times the bats are modified. Pareto points are the number of times the iterations are repeated, and new weights are calculated. Alpha is used to calculate the first objective and loudness of a bat. Beta is used to calculate the first objective and frequency of a bat. Gamma and delta are used to calculate the second objective of a bat. Minimum support and minimum confidence are used to set the minimum acceptable support and confidence values for the mined rules.

In step 1, the bat population is initialized with the population size number of bats, and their objectives are evaluated. The population is sorted based on the objective measure of bats, and the best bat is assigned to the global best variable. The non-dominated solutions list is initialized with an empty list.

Steps 2-9 are repeated pareto points number of times. In step 2, a list of weights used for calculating the objective measure is initialized. Steps 3-7 are repeated iterations number of times.

In step 3, each bat's frequency, velocity, and rule is updated. In step 4, it is checked if a random number is greater than the bat's rate, then one attribute value of the rule is changed according to the global best solution. In step 5, the rule is checked and fixed, and the objectives of the rule are calculated. Step 6 checks if the new rule is better than the old rule, and if it is, the new rule is accepted, and the loudness and rate of the bat are updated.

In step 7, the population is sorted by the objective measure, and the best bat is assigned to the global best variable. In step 9, each bat's best rule is added to the non-dominated rules list. In step 11, the non-duplicated rules from the non-dominated list are returned.

The additional methods in BatAlgorithm are used to initialize the population and generate a bat with a random rule. The implementation includes a method for generating a new solution based on the frequency, velocity, and loudness of a bat. Additionally, methods that check and fix a rule, evaluate objectives, and find the lower and upper bounds are implemented.

Each rule generated by the MOB-ARM implementation is kept in a class called Bat. This class is used to store data about the generated rule. This class has multiple parameters. Frequency and velocity are used to update the velocity of the bat and generate a new

solution with each iteration. Rate is used to determine whether an attribute of the rule should be changed towards the best rule in the population and is increased if a solution is accepted. The initial rate is the first randomly generated rate and is used to update the bat's rate. Loudness is used for generating new solutions. Rule is the generated association rule. Support, confidence, comprehensibility, and interestingness parameters hold the fitness values of the rule. Objective holds the objective measure of the rule.

5 Implementation and experimental setup

The implementation part of this work consists of five parts, which are the helpers used for multiple algorithms, and the implementations of MOPAR, MOCANAR, ACO_R, and MOB-ARM algorithms. The implementations of MOPAR, MOCANAR, ACO_R and MOB-ARM were explained in 4.1.3, 4.2.3, 4.3.3, and 4.4.3, respectively. In this section the additional implementation necessary to run the algorithms is discussed. After that, the dataset, hardware, and parameter descriptions are given.

5.1 Helpers

Several helper functions are necessary to aid the implementation of the chosen algorithms, which are used across multiple algorithms. These functions consist of checking, discretizing, initializing, and querying data. Additionally, calculating fitness functions, and checking for dominance are implemented.

Data checkers are functions that check and fix the locations and attribute values of the generated rules. These methods are important so that after the rules are randomly generated or improved, they can be checked to see if they are valid. For an example, the antecedent and consequent parts of the rule cannot be empty, and these methods check and fix that.

Data discretization functions are used to discretize data into intervals before it can be used in the MOB-ARM algorithm. Other algorithms do not require prior discretization.

Data initialization functions initialize data before it is passed to the algorithms. First, the CSV data is turned into a data frame. After that the lower and upper bounds of each attribute are defined, which is used to ensure that each attribute value of a rule is valid.

Data query functions are the first step to calculating the objective measures for each rule. First the rules are split into antecedent and consequent parts. Then the data frame is queried to find out how many records include the antecedent or consequent part of the

rule, how many records contain both parts of the rule and how many records are in the data frame.

The objective functions are used to calculate the objective measures for the rules, using the numbers returned by the data query functions. Here the support, confidence, comprehensibility, and interestingness measures are calculated and returned.

Other helpers are used to delete the duplicated rules. Rules that have the same antecedent and consequent parts, or the same fitness values, are removed. Additionally, finding non-dominated rules from a population is implemented. A Pareto policy is used for finding non-dominated rules.

5.2 Dataset description

To mine association rules, a dataset is necessary to mine rules from. For the datasets, four datasets from [26] were used. Basketball, Quake, and Bodyfat datasets are the most used for comparison in the papers that proposed the algorithms. An additional dataset, Longley, was chosen. The characteristics of these datasets are shown in Table 1. All the datasets have a different number of records and attributes, which allows for a better comparison of how well the implementations work with different characteristics.

Table 1. Datasets.

Dataset	**Number of records**	**Number of attributes**
Basketball	96	5
Quake	2178	4
Fat	225	18
Longley	16	7

5.3 Hardware description

All tests were executed on an Intel Core i7-10510U machine with 16 GB of memory running under Windows 10.

5.4 Parameter description

This section describes the parameters used for the validation and comparison of the implemented algorithms. The parameters are taken from the papers that proposed them and modified where necessary.

Parameters for the MOPAR algorithm validation are shown in Table 2. These parameters were proposed by [9] as the best-performing parameters for the MOPAR algorithm.

Table 2. MOPAR algorithm parameters for validation.

Population size	**Iterations**	**External repository size**	**Inertia weight**	**Velocity limit**	**xRank**	**C1**	**C2**
40	2000	100	0.63	3.83	13.33	2	2

The MOPAR algorithm comparison parameters are shown in Table 3. The population size, number of iterations and external repository size were set as 50, 200, and 50 for fair comparison, and the other parameters were taken from [9].

Table 3. MOPAR algorithm parameters for comparison.

Population size	**Iterations**	**External repository size**	**Inertia weight**	**Velocity limit**	**xRank**	**C1**	**C2**
50	200	50	0.63	3.83	13.33	2	2

The MOCANAR algorithm validation parameters are shown in Table 4. These parameters were proposed by [12] as the best-performing parameters for this algorithm. However, population size, number of generations and number of increments were decreased because of the time it would have taken to run it with the proposed parameters.

Table 4. MOCANAR algorithm parameters for validation.

Dataset	**Population size**	**Generations**	**Increments**	**Random cuckoo**	**Tournament**	**Pa**	**Pmut**	**W1**	**W2**	**W3**
Basketball	150	150	4	1	30	0.3	0.05	0.2	0.5	0.3
Quake	150	150	4	2	50	0.2	0.2	0.2	0.5	0.3

The MOCANAR parameters for comparison are shown in Table 5. The population size and number of generations were set as 50 and 200 for fair comparison, and other parameters were taken from [12].

Table 5. MOCANAR algorithm parameters for comparison.

Population size	**Gene-rations**	**Incre-ments**	**Random cuckoo**	**Tourna-ment**	**Pa**	**Pmut**	**W1**	**W2**	**W3**
50	200	1	1	30	0.3	0.05	0.2	0.5	0.3

Parameters used for ACO_R algorithm validation and comparison are shown in Table 6. In both cases, the parameters are the same. These parameters were chosen through testing, because [10] did not specify the best parameters for this algorithm. The ant colony size, number of iterations and archive size were set as 50, 200, and 50 for fair comparison.

Table 6. ACO_R algorithm parameters.

Ant colony size	**Iter-ations**	**Archive size**	**Alpha1, alpha2**	**Alpha3, alpha 5**	**Alpha4**	**Q**	**E**
50	200	50	4	1	0.001	0.1	0.85

The MOB-ARM algorithm validation parameters are shown in Table 7. These parameters were proposed by [11] as the best-performing parameters for the MOB-ARM algorithm.

Table 7. MOB-ARM algortihm parameters for validation.

Population size	**Iter-ations**	**Pareto points**	**Alpha**	**Beta**	**Gamma**	**Delta**	**Min supp**	**Min conf**
50	100	10	0.4	0.3	0.2	0.1	0.2	0.5

The parameters used for comparison are shown in Table 8. The population size parameter was set as 50. Number of iterations and pareto points were set as 40 and 5 for fair comparison. Other parameters were taken from [11].

Table 8. MOB-ARM algorithm parameters for comparison.

Population size	**Iter-ations**	**Pareto points**	**Alpha**	**Beta**	**Gamma**	**Delta**	**Min supp**	**Min conf**
50	40	5	0.4	0.3	0.2	0.1	0.2	0.5

6 Analysis

This section tests the implemented algorithms, and the test results are analysed. The implementations are first validated against the papers that proposed the algorithms. After that, the implemented algorithms are compared against each other and analysed.

6.1 Validation

The algorithms were tested and compared against the papers that proposed the algorithms to validate the implementations. The validation was done with one run of tests for all the implementations and datasets.

The MOPAR algorithm validation is shown in Table 9. The number of rules generated using Basketball dataset and the support and confidence using both datasets are similar. The biggest difference is in the number of rules generated using the Quake dataset, which is two times smaller for the implementation.

Table 9. MOPAR algorithm validation.

Dataset	**Source**	**Rules**	**Support**	**Confidence**
Basketball	MOPAR Implementation	67	0.33	0.86
	MOPAR [9]	69.75	0.31	0.95
Quake	MOPAR Implementation	26	0.33	0.80
	MOPAR [9]	54.1	0.32	0.89

The MOCANAR algorithm validation results are shown in Table 10. The implementation was run with a smaller number of increments because of the time it would have taken to run it with the proposed number of increments. This might have affected the results of the implementation and caused the smaller values in the objective measures. However, the results are overall similar. The comprehensibility measure is smaller for the implementation in both datasets, and support is smaller in the Basketball dataset. The number of rules in Quake is bigger for the implementation.

Table 10. MOCANAR algorithm validation.

Dataset	Source	Rules	Support	Confidence	Compre-hensibility	Interest-ingness
Basketball	MOCANAR Implementation	55	0.48	0.82	0.71	0.24
	MOCANAR [12]	55.4	0.66	0.82	0.92	0.38
Quake	MOCANAR Implementation	49	0.53	0.86	0.69	0.28
	MOCANAR [12]	28.2	0.51	0.84	0.95	0.34

The ACO_R algorithm validation is shown in Table 11. The number of rules generated differs a little bit. However, the support and confidence measures are similar.

Table 11. ACO_R algorithm validation.

Dataset	Source	Rules	Support	Confidence
Basketball	ACO_R Implementation	41	0.46	0.79
	ACO_R [10]	37	0.45	0.78
Quake	ACO_R Implementation	46	0.57	0.88
	ACO_R [10]	58	0.60	0.84

The MOB-ARM algorithm validation results are shown in Table 12. The discretization of data was not specified exactly by [11], which might affect the number of rules generated by the implementation, which is several times smaller. The support and confidence, however, are similar.

Table 12. MOB-ARM algorithm validation.

Dataset	Source	Rules	Support	Confidence
Basketball	MOB-ARM Implementation	7	0.31	0.69
	MOB-ARM [11]	63	0.38	0.79
Quake	MOB-ARM Implementation	6	0.50	0.74
	MOB-ARM [11]	50	0.41	0.88

The differences in the validation results are mainly in the number of rules generated and can be attributed to multiple factors. First, the implementations were tested with a smaller number of tests. Additionally, for some of the algorithms, the exact parameters that were tested in the papers that proposed them are unknown or had to be modified. Finally, some steps in the algorithms might not be specified in the proposed papers; thus, the implementations might not work in the same way. However, the values of objective measures are overall similar in all the comparisons.

6.2 Comparison

To find out which algorithm implementation performs the best, the implementations were tested and compared against each other. Five tests were done for each algorithm, using each dataset. The results of the tests were averaged and are shown in Table 13.

Table 13. Comparative results for the implementations.

Dataset	**Algorithm**	**Time (sec)**	**Rules**	**Support**	**Confidence**	**Compre-hensibility**	**Interest-ingness**
Basket-ball	MOPAR	455.4	11.2	0.13	0.78	0.82	0.43
	MOCANAR	404.9	32.8	0.49	0.80	0.67	0.24
	ACO_R	442	40.4	0.41	0.80	0.62	0.25
	MOB-ARM	1181.92	8.4	0.28	0.63	0.62	0.24
Quake	MOPAR	361	18.6	0.22	0.71	0.71	0.16
	MOCANAR	424.04	22.2	0.51	0.84	0.66	0.24
	ACO_R	402.16	47	0.57	0.87	0.63	0.24
	MOB-ARM	1253.42	8.4	0.45	0.72	0.64	0.22
Bodyfat	MOPAR	1259.28	10.4	0.08	0.48	0.83	0.15
	MOCANAR	1469.22	54.6	0.63	0.87	0.69	0.21
	ACO_R	1173.26	13.8	0.01	0.86	0.75	0.54
	MOB-ARM	3345.4	7.8	0.34	0.72	0.62	0.27
Longley	MOPAR	500.28	16.2	0.10	0.94	0.90	0.84
	MOCANAR	545.08	8.8	0.29	0.93	0.75	0.65
	ACO_R	604.52	8.4	0.35	0.99	0.55	0.56
	MOB-ARM	1539.3	20.6	0.28	0.92	0.70	0.59

Figure 10 shows the average time spent by the algorithms. MOPAR, MOCANAR and ACO_R had similar results in all datasets. MOB-ARM was multiple times slower than the other algorithms.

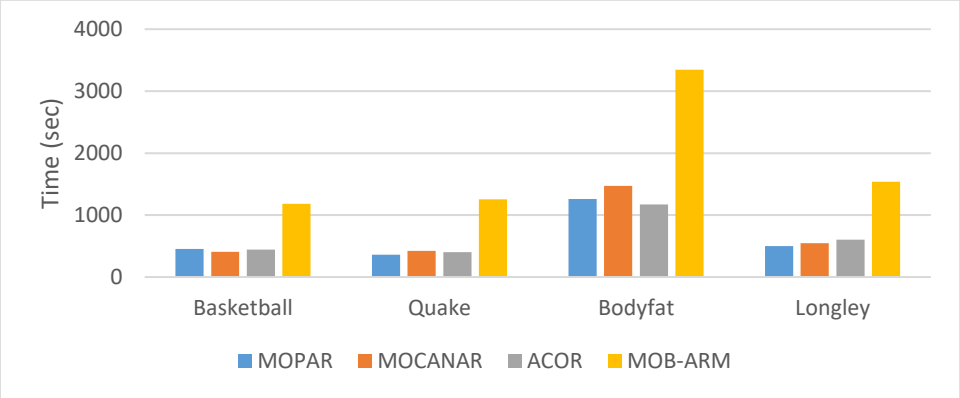

Figure 10. Average time spent by the algorithms.

Figure 11 shows the average number of rules mined by the algorithms. MOCANAR and ACO_R mined the most rules across all datasets. MOPAR and MOB-ARM mined the least rules across all datasets.

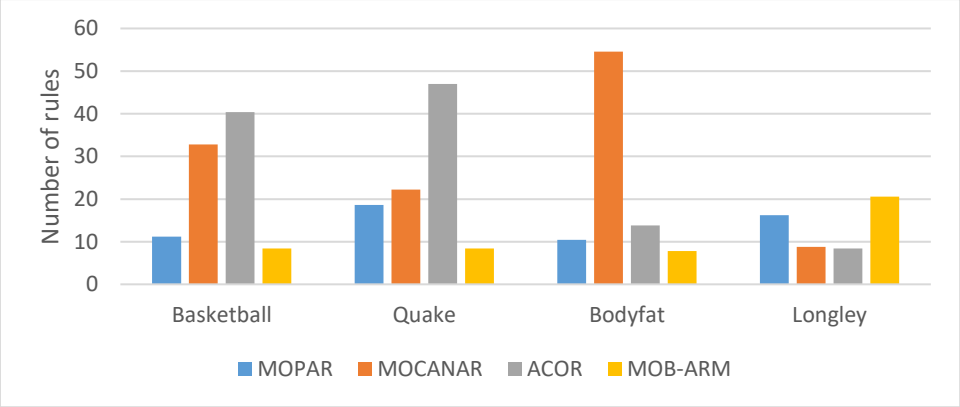

Figure 11. Average number of rules mined by the algorithms.

Figure 12 shows the average support values of the rules mined by the algorithms. MOCANAR had the overall highest results. ACO_R also produced rules with high support in the Basketball, Quake, and Longley datasets but underperformed in the Bodyfat dataset. MOB-ARM had average support measures in all datasets. MOPAR had the overall lowest support values.

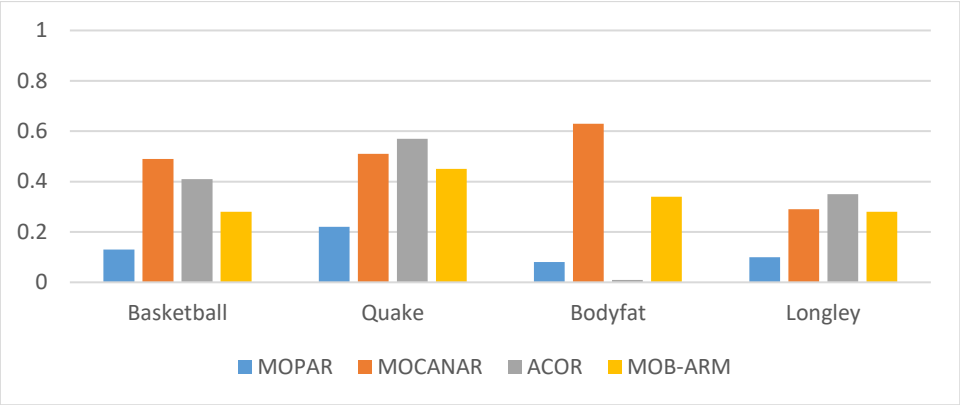

Figure 12. Average support values of the rules mined by the algorithms.

Figure 13 shows the average confidence values of the rules mined by the algorithms. MOCANAR and ACO_R had the highest results across all datasets. MOPAR and MOB-ARM had average results, but MOPAR produced the lowest confidence in the Bodyfat dataset, and MOB-ARM produced the lowest result in the Basketball dataset.

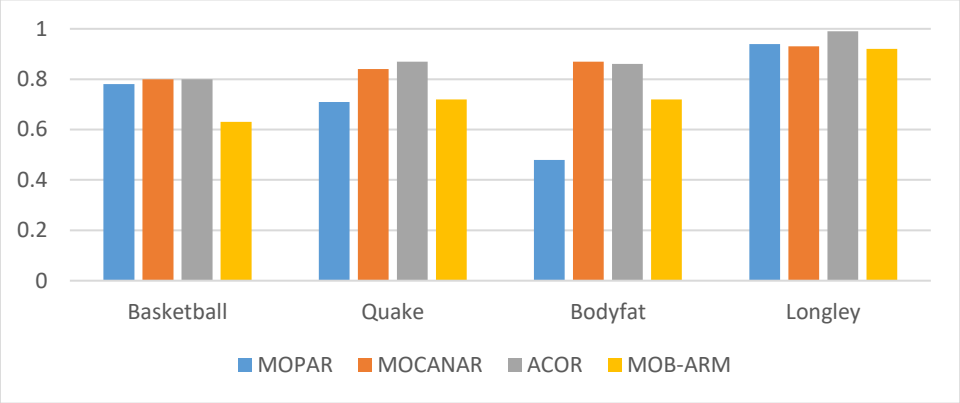

Figure 13. Average confidence values of the rules mined by the algorithms.

Figure 14 shows the average comprehensibility values of the rules mined by the algorithms. MOPAR produced the highest comprehensibility measures in all datasets. MOCANAR, ACO_R and MOB-ARM had similarly average results across all datasets.

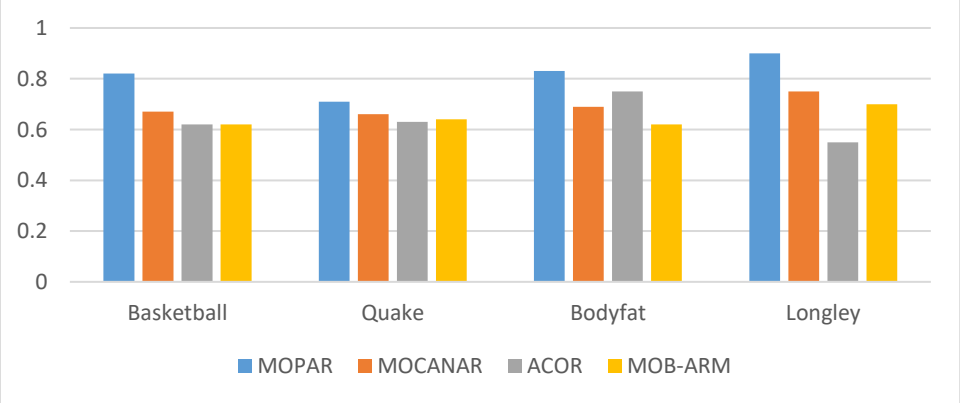

Figure 14. Average comprehensibility values of the rules mined by the algorithms.

Figure 15 shows the average interestingness values of the rules mined by the algorithms. MOPAR produced the highest results in the Basketball and Longley datasets, but the lowest results in the Quake and Bodyfat datasets. ACO_R had the highest interestingness measure in the Bodyfat dataset. MOCANAR and MOB-ARM produced average interestingness results in all datasets.

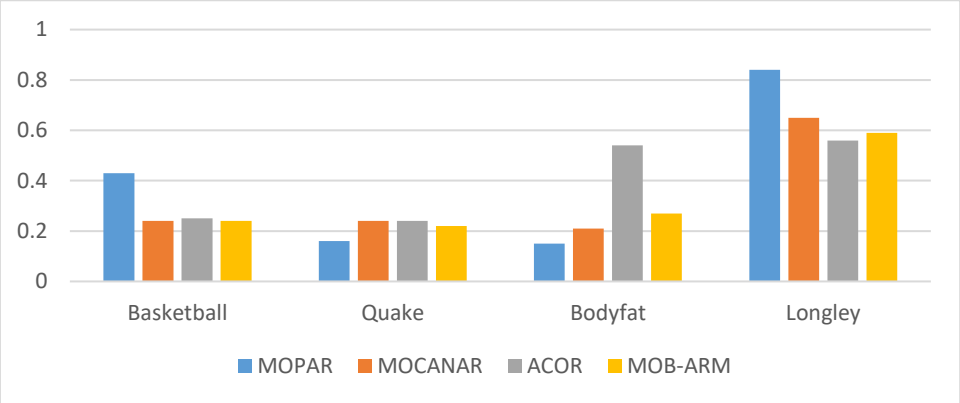

Figure 15. Average interestingness values of the rules mined by the algorithms.

In conclusion, MOCANAR produced the highest and most stable results overall. ACO_R also had high results but underperformed in terms of support in Bodyfat dataset. MOPAR is the best for comprehensibility, but the worst in support measure. MOB-ARM produced the lowest number of rules with average results in all datasets but was the slowest in all datasets.

6.3 Proposed future improvements

The implemented algorithms can be improved in many ways. MOB-ARM algorithm can be improved by modifying, or eliminating the discretization step, to produce more rules. All the algorithms can be improved by making them more efficient, so that they can be better tested with more iterations and bigger datasets.

The validation of the implementations can be improved by using a bigger number of tests to validate the results. The comparison of the implementations can be improved by further testing the algorithms with different parameters to find out which parameters produce the best results for all algorithms and all datasets. It would also benefit the comparison to run a bigger number of tests, because individual test results can vary a lot. Additionally, the comparison can be improved by using a larger number of different datasets.

7 Summary

The main goals of this work were to implement the chosen swarm intelligence-based NARM algorithms and conduct a comparison of the implementations. As a result, the algorithms were implemented according to their descriptions in the papers that proposed them. The implementations were validated against the proposed papers and compared against each other; the results were analysed.

The validation of the implementations showed that the implementations produce association rules that have a similar quality of fitness measures as the papers that proposed them. However, MOB-ARM implementation produced multiple times fewer rules.

The comparison of the implementations showed that different algorithms are suitable for different needs, and each algorithm has its own drawbacks. The MOPAR algorithm can produce a low number of rules with high confidence, comprehensibility, and interestingness measures, but needs parameter modifying when used for datasets with a bigger number of attributes or instances. MOCANAR can be used to produce rules that have stable results in all measures, across all datasets. ACO_R produced high quality rules but needs parameter modification when used for datasets with a bigger number of attributes. MOB-ARM produced a small number of rules with reliable measures in all datasets, because of its minimum support and confidence parameters, but was multiple times slower than other implementations.

The proposed implementations can be improved by modifying or eliminating the discretization step in MOB-ARM, to increase the number of rules generated. The implementations can additionally be made more efficient to decrease the time needed for the algorithms to produce rules. The validation and comparison parts of this work can be improved by further testing the implementations with different parameters and a bigger number of tests.

References

[1] Q. Zhao and S. Bhowmick, "Association Rule Mining: A Survey," Nanyang Technological University, Singapore, 2003.

[2] R. Agrawal, T. Imieli'nski and A. Swami, "Mining association rules between sets of items in large databases," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, Washington, D.C, 1993.

[3] R. Srikant and R. Agrawal, "Fast algorithms for mining association rules," in *Proceedings of the 20th VLDB Conference*, Santiago, Chile, 1994.

[4] J. Han, J. Pei, Y. Yin and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," *Data Mining and Knowledge Discovery,* vol. 8, no. 1, pp. 53-87, 2004.

[5] R. Srikant and R. Agrawal, "Mining quantitative association rules in large relational tables," *ACM SIGMOD Record,* vol. 25, no. 2, pp. 1-12, 1996.

[6] E. V. Altay and B. Alatas, "Intelligent optimization algorithms for the problem of mining numerical association rules," *Physica A: Statistical Mechanics and its Applications,* vol. 540, no. 123142, pp. 1-11, 2019.

[7] M. Kaushik, R. Sharma, S. A. Peious, M. Shahin, S. B. Yahia and D. Draheim, "A Systematic Assessment of Numerical Association Rule Mining," *SN Computer Science,* p. 348, 2021.

[8] I. Fister Jr. and I. Fister, "A Brief Overview of Swarm Intelligence-Based Algorithms for Numerical Association Rule Mining," University of Maribor, Slovenia, 2020.

[9] V. Beiranvand, M. Mobasher-Kashani and A. A. Bakar, "Multi-Objective PSO Algorithm for Mining Numerical Association Rules Without a Priori Discretization," *Expert Systems with Applications,* vol. 41, no. 9, pp. 4259-4273, 2014.

[10] P. Moslehi, B. M. Bidgoli, M. Nasiri and A. Salajegheh, "Multi-Objective Numeric Association Rules Mining via Ant Colony Optimization for Continuous Domains without Specifying Minimum Support and Minimum Confidence," *International Journal of Computer Science Issues,* vol. 8, no. 5, pp. 34-41, 2011.

[11] K. E. Heraguemi, N. Kamel and H. Drias, "Multi-Objective Bat Algorithm for Mining Numerical Association Rules," *Int. J. Bio-Inspired Computation,* vol. 11, no. 4, pp. 239-248, 2018.

[12] I. Kahvazadeh and M. S. Abadeh, "MOCANAR: A Multi-Objective Cuckoo Search Algorithm for Numeric Association Rule Discovery," in *Fourth International Conference on Advanced Information Technologies and Applications*, 2015.

[13] E. V. Altay and B. Alatas, "Performance Analysis of Multi-Objective Artificial Intelligence Optimization Algorithms In Numerical Association Rule Mining," *Journal of Ambient Intelligence and Humanized Computing,* vol. 11, pp. 3449-3469, 2019.

[14] E. V. Altay and B. Alatas, "Association analysis of Parkinson disease with vocal change characteristics using multi-objective metaheuristic optimization," *Medical Hypotheses,* vol. 141, 2020.

[15] E. V. Altay and B. Alatas, "A novel clinical decision support system for liver fibrosis using evolutionary multi-objective method based numerical association analysis," *Medical Hypotheses,* vol. 144, 2020.

[16] M.-S. Chen, J. Han and P. Yu, "Data mining: An overview from a database perspective," *IEEE Transactions on Knowledge and Data Engineering,* vol. 8, no. 6, pp. 866-883, 1997.

[17] C. Blum and X. Li, "Swarm Intelligence in Optimization," in *Swarm Intelligence*, Berlin, Springer-Verlag, 2008, pp. 43-85.

[18] A. Ghosh and B. Nath, "Multi-Objective Rule Mining Using Genetic Algorithms," *Information Sciences,* vol. 163, no. 1-3, pp. 123-133, 2004.

[19] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms, USA: John Wiley & Sons, 2001.

[20] J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *IEEE,* vol. 4, pp. 1942-1948, 1995.

[21] I. Tahyudin and H. Nambo, "The Combination of Evolutionary Algorithm Method for Numerical Association Rule Mining Optimization," *Tahyudin, I., & Nambo, H. (2017). The Combination of Evolutionary Algorithm Method for Numerical Association Rule Mining Optimization. Advances in intelligent systems and computing,* vol. 502, pp. 13-23, 2017.

[22] R. Kuo, M. Gosumolo and F. E. Zulvia, "Multi-objective particle swarm optimization algorithm using adaptive archive grid for numerical association rule mining," *Neural Computing and Applications,* pp. 1-14, 2017.

[23] K. Socha, "Ant Colony Optimization for Continuous and Mixed-Variable Domains," 2009.

[24] X.-S. Yang, "A New Metaheuristic Bat-Inspired Algorithm," *Nature Inspired Cooperative Strategies for Optimization,* vol. 284, pp. 65-74, 2010.

[25] K. Heraguemi, N. Kamel and H. Drias, "Association Rule Mining Based on Bat Algorithm," *Journal of Computational and Theoretical Nanoscience,* vol. 12, no. 7, pp. 1195-1200, 2015.

[26] H. A. Guvenir and I. Uysal, "Function Approximation Repository," Bilkent University, 2000.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Pilleriin Kõiva

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Implementation and Performance Assessment of Swarm Intelligence Based Numerical Association Rule Mining Algorithms” supervised by Minakshi Kaushik
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

30.05.2022

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.