Association Rules Mining by Improving the Imperialism Competitive Algorithm (ARMICA)

S. Ghafari, Christos Tjortjis

Association Rules Mining by improving the Imperialism Competitive Algorithm (ARMICA)

S. Mohssen Ghafari[1], Christos Tjortjis[1*]

[1]School of Science & Technology, International Hellenic
University, 14th km Thessaloniki - Moudania
57001 Thermi, Greece
{m.ghafari, c.tjortjis}@ihu.edu.gr

ABSTRACT. Many algorithms have been proposed for *Association Rules Mining* (ARM), like *Apriori*. However, such algorithms often have a downside for real word use: they rely on users to set two parameters manually, namely minimum *Support* and *Confidence*. In this paper, we propose *Association Rules Mining by improving the Imperialism Competitive Algorithm* (ARMICA), a novel ARM method, based on the heuristic *Imperialism Competitive Algorithm* (ICA), for finding frequent itemsets and extracting rules from datasets, whilst setting support automatically. Its structure allows for producing only the strongest and most frequent rules, in contrast to many ARM algorithms, thus alleviating the need to define minimum support and confidence. Experimental results indicate that ARMICA generates accurate rules faster than Apriori.

Keywords: Association Rules Mining, Data Mining, Knowledge Engineering

1. INTRODUCTION

With the dramatic increase in the amount of available data, searching for information in databases can be done manually with queries, but takes a long time and is not always efficient [1]. Alternatively, Data mining and particularly *Association Rules Mining* (ARM) have been proposed to analyze databases and extract frequent patterns and rules. A well-known method for ARM algorithms employs a two-step approach: find the *Frequent List* and extract *Rules* from that list [2].

However, there are still challenges faced by many ARM algorithms, as they require the user to define two parameters named *Minimum Support* and *Minimum Confidence* [3, 4, 5]. Another challenge for *Apriori* style ARM algorithms is that they use a time and resource consuming process, named *Pruning* which checks the feasibility of a rule and removes weak rules [5]. ARM algorithms could improve by approaches that automatically determine the value for parameters like minimum support and confidence [6, 7, 8], as well as by replacing pruning with more efficient methods. Although there have been researches addressing this topic, more needs to be done.

In this paper, we propose *Association Rules Mining by improving the Imperialism Competitive Algorithm* (ARMICA), a novel dynamic method based on the heuristic *Imperialism Competitive Algorithm* (ICA) [9]. ARMICA customizes itself according to the database that it operates on, and eliminates the dependability on initial parameters like minimum support and confidence. It automatically selects the strongest rules from the database. Moreover, in contrast with algorithms like *Apriori*, it does not involve time-consuming *pruning*. Because it eliminates all the weak items, it results in eliminating the need for pruning. ARMICA improves ICA in that it selects

AIAI'16, September 16–18, 2016, Thessaloniki, Greece.

DOI: http://dx.doi.org/10.1145/12345.67890

imperialists among the most *powerful countries*, rather than randomly, thus alleviating the need for a *revolution* part. In addition, having found the minimum support, ARMICA disqualifies and removes countries that have less *cost* than this minimum support. Finally, ARMICA has a different fitness function and method for creating initial countries. It was evaluated on publicly available datasets against *Apriori*. Experimental results show that it produces accurate results, up to 3 times faster than *Apriori*.

The rest of the paper is organized as follows: Section 2 discusses previous related work. Section 3 introduces and details the *Imperialism Competitive Algorithm* (ICA). Section 4 presents our proposed method, ARMICA. Section 5 presents the evaluation process and Section 6 discusses this work, its contribution and how it complements the state of the art. The paper concludes with directions for future work in Section 7.

2. Related Work

Association Rules Mining (ARM) is a well-known data mining method [2]. It aims at extracting frequent patterns and structure from data. Given T, a set of transactions in a transactional database: $\{t_1, t_2,\ldots, t_n\}$and I, a set of items in this database: $\{i_1, i_2,\ldots, i_n\}$, association rules mining goal is to find all {X, Y} where X and Y are both sets of some items (*itemsets*). In other words, ARM finds all the rules of the form: $X \rightarrow Y$, where X and Y are both subsets of I and $X \cap Y = \Phi$. This rule indicates that whenever X occurs then, Y may occur with certain probability. As a result, extracting such rules could help enrich our knowledge about a database, which supports solving real world problems.

Most of ARM approaches consist of two stages: finding the *Frequent Itemset List* and generating *rules*. One of the best-known ARM algorithms is Apriori [5]. It starts with finding the *candidate Itemset list*. Then, with a technique, named *pruning* and *joining*, it iteratively creates a candidate list and eliminates weak itemsets, employing the concept of Minimum Support. *Support* is the percentage of itemset occurrences in the transactions. Therefore, Apriori uses a user-defined threshold to check itemsets in the candidate list. If there is an itemset that has a support value less than the minimum, it removes it. Moreover, at each level there is a procedure, named pruning, which checks that every itemset in the candidate list satisfies this condition: "all the non-empty subsets of each itemset should be in the previous candidate list". If there is an itemset which does not meet this condition, Apriori removes it from the list. Eventually the candidate list becomes the frequent list. At that point, every non-empty subset of each item in the frequent list can be extracted. The algorithm removes the weaker rules based on a parameter, named Minimum Confidence. The *confidence* of a rule is calculated by:

$$\text{Support } (X \cup Y) \text{ /Support } (X) \tag{1}$$

Therefore, Apriori uses a user-defined threshold for min. confidence removing rules with lower confidence. Apriori is known for its efficiency in extracting frequent itemsets [5], [10]. Its main concept is the *Apriori principle*: any superset of an infrequent itemset must be an infrequent itemset. There were many algorithms that were proposed based on Apriori, which can be classified as follows [11]:

- Candidate itemsets reduction:

A smaller number of candidate itemsets incurs reduced computational cost. The hash-based DHP is such example [12]. Its main advantages are: efficient generation of frequent itemsets, effective declining in transaction database size and reduction in number of database scans. Before the next database scan, it uses a hash table to reduce the size of the candidate k-itemsets (that is C_k, for k>1). The number of candidate k-itemsets may be reduced substantially, especially when the length of itemsets is equal to 2. However, DHP has a similar to Apriori behavior, as it needs to scan the database as many times as the length of the longest frequent itemset in it.

- Database scans reduction

A large number of database scans increases the cost of time-consuming disk I/O. In order to tackle this problem, Savasere et al. proposed the Partition algorithm [13], which produces all frequent itemsets with two database scans. The Partition algorithm divides the database into several chunks in such a way that each portion can fit in memory and can be processed by Apriori. However, one of its drawbacks is that the number of candidate itemsets examined is much bigger than that of Apriori. The Dynamic Itemset Counting (DIC) algorithm [14], also divides the database into several parts. Once some frequent itemsets are produced, DIC can generate new candidate itemsets and immediately starts counting them. It should also be noted that the data distribution of a database has great impact on the performance of DIC [14].

- Combination of bottom-up and top-down search

This algorithm also employs the Apriori principle to identify frequent itemsets in the bottom-up search, while in each round the infrequent itemsets found in the same direction are used to separate the maximal frequent candidate itemsets in the top-down direction. The advantage is that all subsets of the maximal frequent itemsets are ready as long as the maximal frequent itemsets are identified. As a result, there is no need to extract subsets of the maximal frequent itemsets again from the bottom-up direction. The Pincer-Search algorithm [15] and the MaxMiner algorithm [12] are two examples of this concept, but they are not efficient in finding the maximal frequent candidate itemsets. Meanwhile, the advance made by this approach is not remarkable when the length of the longest frequent itemset is relatively short [11].

- Proposing Dynamic ARM Algorithms

Researchers proposed dynamic algorithms to deal with the drawbacks of setting the minimum support and confidence manually. For instance, Djenouri et al. proposed HBSO-TS, a method that uses two heuristic algorithms: Artificial Bee Colony (ABC) for the main processing and Tabu Search (TS) for searching the neighbors of each food source for each bee [6]. The authors concede that setting many parameters required by ABC and TS is hard. In addition, they do not mention anything about possible advantages of their method concerning pruning.

Kuo et al. [7] propose an algorithm that tries to determine minimum support and confidence automatically. It uses a heuristic algorithm, named Particle Swarm Optimization (PSO). First, they apply particle swarm encoding, which they claim to be similar to chromosome encoding of genetic algorithms. Next, they calculate the fitness value, and, based on that, they generate a population of particle swarms. Finally, PSO searching is applied until it satisfies the stop condition that is finding the best particle. The support and confidence of the best particle can represent the minimal support and minimal confidence. It should be noted that setting the best solution as minimum support may not be efficient. Besides, they use minimum confidence to evaluate their final rules, like most ARM algorithms.

Nandhini et al. [8] propose an algorithm also based on PSO and a domain ontology. Like PSO, it determines min. support automatically. It also considers support and confidence of the best particle as minimum support and confidence. However, it also needs to set the minimum confidence in order to evaluate rules. Next, we detail the Imperialism Competitive Algorithm (ICA).

3. Imperialism Competitive Algorithm (ICA)

ICA is a powerful heuristic algorithm based on the notion of the relation between imperialistic countries and their competition for acquiring colonies [9]. ICA involves key concepts that we present and discuss here:

- *Country*: the variables that we want to optimize
- *Imperialist*: A powerful country that can control other countries in the form of colonies.
- *Colony*: less powerful country that imperialists try to control.
- *Empire*: a group of countries headed by an imperialist.
- *Cost*: a parameter depending on the nature of the problem at hand. In this paper, we consider it the number of occupancies of item in all the transactions.
- *Normalized cost*: the cost of each country calculated using formula (3) below.
- *Power*: a factor calculated based on the cost of each country. A country could be considered as more powerful than another, depending on their costs.
- *Total empire Power*: The sum of the powers of the imperialist and its colonies in one empire.
- $N_{Country:}$ the number of all countries (items)
- N_{imp}: the number of Imperialists, which is a parameter [16].
- N_{col}: the number of colonies given by: (Number of all countries – number of Imperialists)

ICA firstly considers all the variables that we want to optimize, as a concept named country. Then it randomly selects some countries as Imperialists. Then it calculates the cost of imperialists and calculates their powers. The cost of each country could be different depending on the case. For instance, in this paper we consider each item as a country and the number of occurrences of each item in all the transactions as the cost of each country. Then, based on the power of imperialists, it divides other countries (colonies) among imperialists. In this step, the algorithm assumes that each imperialist and its colonies are part of an Empire.

Next, ICA calculates the power of each empire based on the power of its colonies and imperialist and establishes a competition between empires. In this step, the most powerful empires remove colonies from the weakest empires. If there is only one colony left for the weakest empire, then the imperialist and the last colony of the weakest empire become colonies of the strongest empire. This competition continues until there is only one empire left, as shown in Table I.

3.1 Creating the Initial Empires

ICA considers an array of variables (*countries*) which should be optimized. We calculate the *cost* of each country by a function *f* at variables $(p_1, p_2, p_3, ..., p_{N_{\text{var}}})$ [9]:

$$cost_i = f(country_i) = f(p_1, p_2, p_3, ..., p_{N_{\text{var}}}) \tag{2}$$

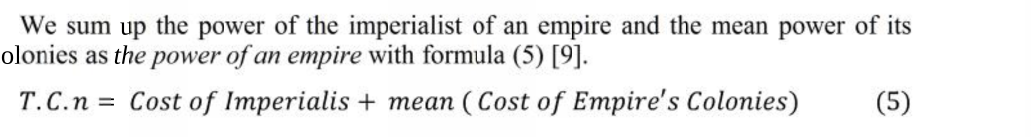
At the beginning, ICA initializes countries of size N_{pop} and randomly selects N_{imp} the countries as initial imperialists. It also considers all the remaining countries (N_{col}) as potential colonies. They should be distributed across the imperialists, based on their power. The definition of *normalized cost* of an imperialist is defined in [16]. We apply this formula to calculate the costs of Items:

$$C_n = \max_i \{c_i\} - c_n \qquad (3)$$

where c_n is the cost of the nth imperialist and C_n is its normalized cost. After having calculated the normalized cost of all imperialists, the normalized power of each imperialist can be computed as follows [16]:

$$p_n = \left| \frac{C_n}{\sum_{i=1}^{N_{imp}} C_i} \right| \qquad (4)$$

3.2 Total Empire Power



We sum up the power of the imperialist of an empire and the mean power of its colonies as *the power of an empire* with formula (5) [9].

$$T.C.n = \mathit{Cost\ of\ Imperialis} + \mathit{mean}\ (\mathit{Cost\ of\ Empire's\ Colonies}) \qquad (5)$$

TABLE I. **IMPERIALISM COMPETITVE ALGORITHM**

1) Initialize the empires
2) Calculate the total Power of the empires.
3) Competition between empires. Select a colony from the weakest empires and assign it to the most powerful one.
4) Eliminate the weakest empires.
5) If there is only one empire left, stop the process. Otherwise, go to step2

4. Proposed Method

Section 3 introduced the original ICA algorithm. This section discusses our proposed method, ARMICA, which employs ICA, suitably modified for association rules mining. Some parts of original ICA are not suitable for ARM, thus we had to change them. For that reason, we modified ICA to adapt it with the characteristics of an ARM algorithm. Firstly, we select imperialists among the most powerful countries, in contrast to ICA, which selects them randomly. Next, we remove the revolution part of ICA. Revolution is about checking if a colony becomes more powerful than the imperialist of its empire is. In that situation, a revolution takes place and ICA exchanges that colony with the imperialist. However, in ARMICA, since we select the weakest colonies and imperialists are the most powerful countries, there is no chance for a colony to become more powerful than its imperialist. As a result, ARMICA does not need incorporate revolution. In addition, having found the minimum support in the first run of ICA, we are using this value to disqualify countries that have less cost than this minimum support. This helps to remove many unusable countries; there is no such mechanism in the original ICA. Finally, there are other modifications, which were necessary for ARMICA in order to improve ICA, like changing the fitness function and creating the initial countries. Table II illustrates pseudo code for ARMICA.

TABLE II. PSEUDO CODE FOR ARMICA

```
Input: Dataset     Output: Rules
1) Run ICA with the dataset and create initial countries based on items (attributes)
2) Calculate the cost for each country
3) Create the empires and calculate the power of each empire
4) Establish competition among empires based on their power
5) The Candidate List consists of the imperialist of the last empire and its colonies
6) In the first candidate list, consider the cost of each country as its support and then sort the
   countries based on their support. Select the median support as the primary min support for
   the next steps.
7) While there is at least one country that has support more than that primary min support do:
   Extract all the combinations of countries and store them in a list, send it again to ICA and
   get the next Candidate List
8) The last candidate list is the Frequent List
9) Extract all the subsets of items in the FrequentList and send them to ICA to get the best
   rules.
```

As Table II indicates, ARMICA uses the dataset as input and delivers association rules as output. In the first step, it creates a set of initial countries based on dataset attributes (initial countries). In other words, it defines each item as a country. As a result, there are as many items (attributes) in the dataset as there are countries for ICA. At this point, the algorithm calculates the cost for each country based on the number of occurrences of each item in the set of transactions. ICA considers countries with lower cost as more powerful.

However, in this paper, since we consider the cost of a country as the number of its occurrences in the set of transactions, it is more suitable that the countries with higher costs are items that are more desirable. As a result, in contrast to ICA, ARMICA assumes that the most powerful countries are those with the highest cost. After calculating the cost for all the empires, the algorithm selects the most powerful countries as imperialists and considers the remaining countries as colonies. Hence, imperialists are the most frequent items in the dataset. The number of Imperialists is a free parameter in the original ICA, but for ARMICA, after experimentation, we consider it the number of countries divided by 10. In addition, if the initial number of countries is $N_{Country}$, and we select N_{imp} countries as imperialists, we have N_{Col} colonies.

In the next step, ARMICA calculates the power of each imperialist and distributes the colonies (other countries or other items) among them based on their power. As a result, by gathering imperialists and colonies together, the initial empires are built. ARMICA calculates the power of each empire based on the sum of powers of its colonies and their imperialist. For this purpose, the algorithm calculates the number of occurrences of each item in all the transactions, considers this value as the cost of that item (country), and calculates its power.

Next, ARMICA establishes a *competition* between empires, where the more powerful empires “steal” colonies from the weakest ones. Another difference between ARMICA and ICA is that in the original algorithm, each colony removed from the weakest empire is placed in the most powerful empire. However, ARMICA removes the colonies and stores them in a list, named *Reserve List*. If there is only one colony left in the weakest empire, the colony and its imperialist, become members (colonies) of the most powerful empire. This procedure continues until all the empires are eliminated and only the most powerful one remains. For this purpose, ARMICA removes the weakest colonies (items) of the weakest empire and keeps them in the *Reserve List*.

If the weakest empire has only one colony, then ARMICA removes that empire and considers its imperialist and its last colony as colonies of the most powerful empire. After several iterations, all the empires will be eliminated and only one empire remains. Now, if there is a member of the Reserve List which is more powerful, i.e. it has higher frequency than any colony in the last empire, the algorithm exchanges them. Hence, members of the final empire constitute our candidate list.

Overall, even though ARMICA uses a minimum support threshold, it sets it automatically and removes the weaker items. The procedure for setting the minimum support involves storing support values, which is the cost (frequency) of each country (item) in the dataset, for all the items of the first generated candidate list and sorting them. At that point, the algorithm considers the median support of all the countries as the universal minimum support. This value is also used to remove countries that have less cost than the minimum support in the next levels.

Then, ARMICA extracts all the combinations of items, stores them in a list, and sends them to ICA. This continues until there is at least one item with support higher than the minimum support**.** As a result, each time the algorithm produces an updated candidate list. It is noticeable that, here is another significant improvement compared to Apriori. In Apriori there is a step, named Pruning which eliminates itemsets with at least one of their subsets is not present in previous candidate lists. As it is clear, this process also needs to calculate all the subsets of the items and check their existence in the previous candidate list. In ARMICA, no such step is needed, as the algorithm selects the most powerful (frequent) items, thus eliminating the need for pruning.

Finally, the last candidate list becomes the Frequent List. At that point, ARMICA extracts all the non-empty subsets of all the items and sends them to ICA. If “I” is the item, then for every nonempty subset of s, we have a rule as:

s → (I-s), in the original Apriori, we have a condition:

Support (I) / Support (s) >= Minimum Confidence (6)

If a rule satisfies this condition, it is selected as a frequent rule. However, since the members of the last empire in ICA are the most powerful rules (have more confidence compared to other rules), ARMICA only selects the most powerful (frequent) subsets and returns the most powerful rules. Again, it is worth mentioning that, ARMICA does not need to have a minimum confidence parameter. Apriori and many other ARM algorithms need a user defined fixed threshold or even a dynamic threshold for minimum confidence. However, our approach only selects the most powerful rules among all the rules and removes all other rules.

4.1 Example

In this section, we explain our algorithm using a small example. Table III illustrates a set of six transactions and eight items.

TABLE III. TRANSACTION DETAILS

	I1	I2	I3	I4	I5	I6	I7	I8
T1	t	t	t	t		t		t
T2		t	t	t		t		
T3	t		t				t	
T4	t		t	t		t		t
T5		t						t
T6	t	t	t		t		t	t

T_1, T_2,…, T_6 are 6 transactions and I_1, I_2, …, I_8 are items in these transactions. First, the algorithm selects some of the most powerful countries as Imperialists. Since there are only 8 items, if we do not use just one imperialist (given by: number of countries / 10) as there would be no competition resulting in selecting all the items, but instead assume that there are 3 imperialists. Their power is calculated and the remaining countries are distributed among the imperialists based on their power:

Imperialist1: I_3	
I_1	I_7

Imperialist2: I_2	
I_4	I_6

Imperialist3: I_8
I_5

Initial empires are now built. ARMICA calculates the cost of countries and the power of each empire based on the sum of the powers of its colonies plus its imperialist. In this occasion, we have:

Empire1's Power: 11(Powerful Empire), Empire2's Power: 10, Empire3's Power: 5 (Weakest Empire)

Then, the algorithm establishes a *competition* between empires. As a result, a powerful empire tries to "steal" colonies from the weakest empire. Finally, if that weak empire has only one colony left, the imperialist and colony of that empire become members of the powerful empire and the algorithm eliminates that empire. As a result, Empire 3 is eliminated and I_8 and I_5 now belong to Empire 1.

Imperialist1: I_3			
I_7	I_1	I_5	I_8

Imperialist2: I_2	
I_6	I_4

The competition goes on. This time Empire 1 tries to get one of the colonies from Empire 2. I_6 is removed from Empire 2 and saved in the *Reserve List*. Since Empire 2 has only 2 members, it joins Empire 1.

Imperialist 1: I_3					
I_7	I_1	I_2	I_8	I_4	I_5

Reserve List
I_6

Now the members of the Reserve List are compared with the Last Empire colonies. If there is a colony that has lower cost than the members of the reserve list do, then they are replaced. In this case, I_6 has higher cost compared to I_5, so the algorithm replaces them and removes I_5. Clearly, ARMICA did not use minimum support and automatically eliminated weak countries. In this step, the algorithm considers the last empire as the Candidate List. At that point, it sorts the items of the candidate list based on their cost and selects the median cost of them as Minimum Support. As a result, the cost of the I_7 becomes the minimum support, that is 4.

Next, the algorithm joins all the countries in a candidate list and calculates their costs. It is noticeable that the cost of a country like I_1I_2 is calculated by the number of occurrences of I_1 and I_2 in the same transaction, that is 2. The algorithm follows the same pattern that we mentioned before, repeatedly until there is no item left in the produced candidate list that has an equal or higher support (cost) than the minimum support. In this situation, the previous candidate list is the Frequent List. Clearly, ARMICA does not require pruning.

Frequent List: I_1I_3

Finally, it is time to extract all the non-empty subsets of the frequent list items and after that send them to ICA in order to find countries that are more powerful. Like previously, the last empire consists of the most powerful countries that represent the strongest rules. It is clear that we did not need to define Minimum Confidence. The last empire, whose members are the final rules, is here:

I_1	→	I_3
I_3	→	I_1

TABLE IV. EVALUATION CHARACTERISTICS

Data set	Items	transactions	Algorithm	Predefined Min. Support	Predefined Min. Confidence	Generated Rules	time (sec)
Supermarket	217	4627	Apriori	27.5 %	71 %	104	17
			ARMICA	—	—	104	6
Breast Cancer	10	286	Apriori	36.18%	47%	75	0,78
			ARMICA	—	—	75	0,76
Nursery	8	12960	Apriori	33.33%	1	2	1.22
			ARMICA	—	—	2	1.15

5. Evaluation

In this section, we evaluate ARMICA, the proposed method. We implemented it using Java 1.7 in Netbeans IDE 7.2 and ran it on an Intel (R) Core (TM) i5 CPU at 2.40 GHz and 4 GB RAM. We also used the implementation of Apriori from Weka 3.6 [17] along with the Supermarket, Breast Cancer and Nursery datasets from the UCI Machine Learning Repository [18] to benchmark our method. Supermarket has 217 attributes and 4627 transactions, Breast Cancer has 10 attributes and 286 transactions, whilst

Nursery has 8 attributes and 12960 transactions. We considered two factors for this evaluation: accuracy and execution time. An improvement on Apriori should decrease the execution time without decreasing accuracy. ARMICA would be a good approach, if it produced accurate rules faster.

The minimum support that ARMICA determined automatically was 27.5%, 36.18% and 33.33% for the Supermarket, Breast Cancer and Nursery datasets, whilst minimum confidence was 71%, 47% and 1%, respectively. We used the same values for Apriori. Table IV illustrates the characteristics of the evaluation process. Apriori and ARMICA produced the same rules. In addition, we compared the two algorithms concerning their execution time. ARMICA took 6 sec compared to Apriori's 17 sec for Supermarket and 0,76 sec compared to Apriori's 0,78 for Breast Cancer. Finally, it required 1.15 sec for Nursery whereas Apriori required 1.22 sec. This means that ARMICA generated the same rules as Apriori, but in a shorter period.

6. Discussion

Many ARM algorithms need two parameters to be defined in advance by the user, who has no guidelines on selecting appropriate values, thus resulting in a trial and error exercise or guesswork. These parameters are used to eliminate infrequent itemsets and weak rules. With the increasing amounts of data available, it is not effective to use an algorithm that requires users to define these parameters. In this paper, we proposed an algorithm, named ARMICA, which applies the ICA algorithm, suitably modified, to find the most frequent rules of a dataset. In contrast to many algorithms, ARMICA sets minimum support automatically and does not need a minimum confidence parameter in order to eliminate the weakest rules.

ARMICA uses ICA, which is a heuristic algorithm to analyze a database. For this purpose, at the beginning, it produces the first candidate list. At that point, it considers the minimum support of itemsets in that list as the global *Minimum Support*. After that, it joins itemsets and produces a new candidate list. This process continues until there is no item left in the candidate list that has support that is equal or more than the Minimum Support. Next, ARMICA considers the last candidate list as Frequent List, extracts all the non-empty subsets of the items, and sends them to ICA. ARMICA return the strongest rules without requiring minimum confidence setting. It is worth mentioning that, since ARMICA sends all the extracted subsets to ICA, ICA only returns subsets that according to formula (1) has higher confidence.

Another advantage of the proposed method is that instead of having a time consuming step, like Pruning, which involves extracting all the subsets of all the items, and trace their existence in the previous candidate list, ARMICA selects the most valuable items. Experimental results indicate that ARMICA has up to 3 times less execution time than Apriori in the Supermarket dataset and shorter execution time in Breast Cancer and Nursery datasets. However, since each improvement in execution time of Apriori may cause some decreasing in accuracy of generated rules, we also considered the accuracy of the ARMICA. The proposed method generates all the rules that Apriori generates. Hence, there is not any accuracy reduction in ARMICA.

7. Conclusion and Future Work

Data mining has attracted attention partly because of the enormous amount of data available. A well-established approach for extracting frequent rules and patterns from data is Association Rules Miming (ARM). Many algorithms have been proposed in this area [1, 2, 3, 4, 5, 19, 20]. However, there are still areas that need to be investigated further.

In this paper, we proposed a new approach, named ARMICA, based on the Imperialism Competition Algorithm. It defines minimum support automatically and extracts the best rules without the need to define minimum confidence, either. In addition, it does not need a pruning stage, as the one used by many Apriori-like ARM algorithms. However, it requires setting of the number of imperialists, a free parameter in the original ICA. We explored ways to calculate it, and found empirically that the number of countries, divided by 10, is a sensible solution, which does not require domain knowledge nor details about the dataset. In the future, we envisage optimizing this selection, alleviating the need for the user to pre-define this parameter.

Finally, ARMICA produces accurate rules fast. In the other words, it generates all the rules that Apriori produces, in a shorter time. Initial experiments have indicated that it is also faster than FP-growth, but proper evaluation is required. We plan to focus on other potential capabilities of ARMICA such as dealing with massive and/or dynamic data sets, benchmark it against state of the art ARM algorithms and investigate performance related areas, such as memory usage and time consumption.

Acknowledgements

The authors thank the Hellenic Artificial Intelligence Society (EETN) for covering part of their expenses to participate in AIAI 16.

REFERENCES

1. Bhandari A., Gupta A., Das D: Improvised apriori algorithm using frequent pattern tree for real time applications in data mining. In: *Proc. Int’l Conf. Information Communication Technologies* (ICICT), Vol. 46, pp. 644–651, (2014).
2. Qodmanan H.R., Nasiri M., Minaei-Bidgoli B.: Multi objective association rule mining with genetic algorithm without specifying minimum support and minimum confidence. In: *Expert Systems with Applications (Elsevier)*, Vol. 38 Issue 1, pp. 288-298, (2011).
3. Dong. L., Tjortjis. C.: Experiences of Using a Quantitative Approach for Mining Association Rules. In: *(Springer) Intelligent Data Engineering and Automated Learning*, Vol. 2690, pp. 693-700, (2003).
4. Wang. C., Tjortjis. C.: PRICES: An Efficient Algorithm for Mining Association Rules. In: *(Springer) Intelligent Data Engineering & Automated Learning*, Vol. 3177, pp. 352-358, (2004).
5. Agrawal, R., and Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: *VLDB ’94 Proc. of the 20th Int’l Conf. on Very Large Data Bases*, pp. 487-499, (1994).
6. Djenouri, Y., Drias, H., and Chemchem, A.: A Hybrid Bees Swarm Optimization and Tabu Search Algorithm for Association. In: *IEEE World Congress on Nature and Biologically Inspired Computing* (NaBIC), pp. 120–125, (2013).

7. Kuo, R., Chao, C., and Chiu, Y., 2011. Application of particle swarm optimization to association rule mining. In *Applied Soft Computing*, vol. 11, pp. 326–336 (January 2011).

8. Nandhini J.M., Janani, M., Sivanandham, S.N.: Association rule mining using swarm intelligence and domain ontology. In: *IEEE Int'l Conf. on Recent Trends in Information Technology (ICRTIT)*, pp. 537-541, (2012).

9. Atashpaz-Gargari, E., and Lucas, C.: Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In: *IEEE Congress on Evolutionary Computation*, pp. 4661 - 4667, (2007).

10. Agrawal, R., Imielinski, T., and Swami, A.: Mining association rules between sets of items in large databases. In: *ACM SIGMOD Conf. on Management of Data*. pp. 207-216, (1993).

11. Yang, D.L., Pan, C.T., and Chung, Y.C.: An Efficient Hash-Based Method for Discovering the Maximal Frequent Set. In: *Proc. 25th Annual Int'l Computer Software and Applications Conf., pp. 511– 516, Chicago* (2002).

12. Park, J. S., Chen, M. S., and Yu, P. S.: An Effective Hash Based Algorithm for Mining Association Rules. In: *Proc. 1995 ACM SIGMOD Int'l Conf. on Management of data*, pp. 175-186, (1995).

13. Savasere, A., Omiecinski, E., and Navathe, S.: An Efficient Algorithm for Mining Association Rules in Large Databases". In: *VLDB '95 Proc. 21th Int'l Conf. on Very Large Data Bases*, pp. 432-444, (1995).

14. Brin, S., Motwani, R., Ullman, J. D., and Tsur, S.: Dynamic Itemset Counting and Implication Rules for Market Basket Data. In: *ACM SIGMOD Conf. on Management of Data*, pp. 255-264, (1997).

15. Lin, D., and Kedem, Z. M.: Pincer-Search: A New Algorithm for Discovering the Maximum Frequent Set. In: (Springer) *VI Intl. Conf. on Extending Database Technology*, Vol. 1377, pp. 103-119, (1998).

16. Talatahari, S., Farahmand, A. B., Sheikholeslami, R., and Gandomi, A.: Imperialist competitive algorithm combined with chaos for global optimization. In: *Communications in Nonlinear Science and Numerical*, Vol. 17, Issue 3, pp. 1312–1319, (2012).

17. Witten, I.H., Eibe, F., and Hall M.A., *Data mining: practical machine learning tools and techniques*. 3rd ed., Morgan Kaufmann, (2011).

18. Bache, K. and Lichman, M. *UCI machine learning repository*, (2013).

19. Han J., Pei J., Yin Y.: Mining frequent patterns without candidate generation. In: *ACM SIGMOD int'l conf. on Management of data*, pp. 1-12, USA (2000).

20. Scheffer T.: Finding Association Rules That Trade Support Optimally against Confidence. In: *Principles of Data Mining and Knowledge Discovery*, Vol. 2168, pp 424-435, (2001).