ARMICA-Improved: A New Approach for Association Rule Mining

Shahpar Yakhchi[1(✉)], Seyed Mohssen Ghafari[1], Christos Tjortjis[2], and Mahdi Fazeli[3]

[1] Computer Science Department, Azad University, Borujerd, Iran
computermsc.y@gmail.com, mohssenghafari@gmail.com
[2] School of Science and Technology Department, International Hellenic University, Thermi, Greece
c.tjortjis@ihu.edu.gr
[3] Computing Department, Iran University of Science and Technology, Tehran, Iran
m_fazeli@iust.ac.ir

Abstract. With increasing in amount of available data, researchers try to propose new approaches for extracting useful knowledge. Association Rule Mining (ARM) is one of the main approaches that became popular in this field. It can extract frequent rules and patterns from a database. Many approaches were proposed for mining frequent patterns; however, heuristic algorithms are one of the promising methods and many of ARM algorithms are based on these kinds of algorithms. In this paper, we improve our previous approach, ARMICA, and try to consider more parameters, like the number of database scans, the number of generated rules, and the quality of generated rules. We compare the proposed method with the Apriori, ARMICA, and FP-growth and the experimental results indicate that ARMICA-Improved is faster, produces less number of rules, generates rules with more quality, has less number of database scans, it is accurate, and finally, it is an automatic approach and does not need predefined minimum support and confidence values.

Keywords: Association rules mining · Data mining · Imperialist Competitive Algorithm (ICA)

1 Introduction

Association Rules Mining (ARM) is a data mining technique to extract frequent rules and patterns from a database. Many challenges are still remained in ARM techniques. Being a single dimension solution and focusing only on one aspect of ARM problems is the main drawback of these methods. For instance, many of them tend to be a fast approach and do not consider other parameters like the number of generated rules or accuracy; or only focus on accuracy of generated rules and do not investigate other factors like being a fast approach or having the least number of database scans. In conclusion, an efficient ARM approach should consider all these parameters at the same time.

G. Li et al. (Eds.): KSEM 2017, LNAI 10412, pp. 296–306, 2017.
DOI: 10.1007/978-3-319-63558-3_25

One of the main parameters that have been frequently considered in many ARM approaches is having low execution time. Generating frequent and interesting rules in a short period of time is one of the primary goals of many ARM approaches. To be a fast approach, many researchers have worked on other parameters that may affect the execution time, like the number of database scans or the number for generated rules. In contrast, in many cases generating accurate rules or even rules with more quality, in this paper we assume that rules with more confidence are more qualify, is in higher priority compared to having low execution time. Finally, an automatic ARM method could be independent from user knowledge and can be applied on any databases. For this reason, some researchers have worked on making their approaches automatic.

In this paper, we focused on different aspects of ARM at the same time. We proposed a new ARM approach, named ARMICA-Improved, which extracts frequent rules accurately and in a short period of time. This approach is based on a heuristic algorithm called Imperialist Competitive Algorithm (ICA) [1]. ARMICA-Improved scans the database only once and generates less number of rules compared to the well-known ARM approaches. It does not consider infrequent items and only selects the most frequent items. In addition, it eliminates the transactions that do not contain any of these frequent items. Finally, our approach is an automatic approach and set the minimum support automatically and does not need minimum confidence to extract frequent rules. The experimental results indicate that ARMICA-Improved has lower execution time, less number of database scans, less number of generated rules, set the minimum support automatically and does not need minimum confidence value; also its generated rules are accurate, and generating more qualify rules compared to the Apriori [2, 3] and FP-growth [4].

The rest of this paper is organized as follows. Section 2 reviews the literature. Description of our proposed method and an example could be found in Sects. 3 and 4, respectively. Our experimental results would be in Sect. 5. Section 6 discusses the experimental results and there would be a conclusion statement in Sect. 7.

2 Literature Review

Apriori [2, 3] is the most famous ARM. Many algorithms tried to improve Apriori whilst others follow different approaches compared to Apriori. Apriori's mechanism is as: T = {t1, t2,…, tn} and I = {i1, i2,…, in} are the set of transactions and set of items that this dataset has, respectively. The algorithm tries to find all {X,Y} that both X and Y may contain at least one item. The extracted rule may be like: $X \rightarrow Y$

This rule means that if we find X in a transaction, then with a probability (Confidence) we also find Y in that transaction. The important thing is X and Y should not have any item in common: $X \cap Y = \Phi$

Most of ARM algorithms have two steps: first, finding the frequent itemsets. Frequent itemsets are sets of items that frequently occur together in the database. Secondly, generate frequent rules from the frequent itemsets. In Apriori, there is a parameter, named minimum support that items and itemsets with frequency of more than minimum support are frequent. Support of each item is the number of occurrence of that item in the database. In each level, Apriori generates candidate list of frequent

items and itemsets. Then, it removes the items and itemsets with support of lower than minimum support. In this step, the algorithm employs a technique, named pruning to check that is there any itemsets, which has an item that was not on the candidate list in the previous levels; if it find one, so it removes this itemset. After pruning, it joins all the items and itemsets in the candidate list with each other and produces new candidate list. This process will continue and Apriori generates 2-length, 3-length, 4-length,… itemsets. It is worth to mention that, in this algorithm user should set the minimum support in advance and manually.

Finally, the last candidate list is the frequent list. At that point, Apriori extracts all non-empty subset of each item generates rules. In this step, Apriori needs another user defined parameter, named Minimum Confidence. Based on that, the algorithm removes the weak rules. Confidence of each rule could be calculated by:

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

In the literature, many heuristic approaches have been proposed. One of heuristic approaches in ARM is [5]. Authors have proposed two new ARM algorithm based on GA, named IARMGA and Memetic algorithm, named IARMMA. They claim that most of bio-inspired-based algorithms have two main drawbacks: Generating false rules and considering some rules with low support and confidence as high qualify rules. They considered two parameters to evaluate their approaches and compared them with each other: Execution time and Accuracy of generated rules. Accuracy in this approached has been considered as value of their fitness function. For this reason, they propose a new method, named "delete and decomposition strategy" to have better accuracy. Finally, their experimental results indicate that IARMMA has higher execution time compared to IARMGA especially in a big dataset. However, IARMMA has better solution quality. In the end, they claimed that their approaches solved the problems of generating false and inaccurate rules. The main drawback of this work may be lack of comparison with other famous methods like Apriori.

Yan et al. have proposed a novel approach based on Genetic algorithms for ARM [6]. In their method, they did not use any fixed minimum support threshold. Instead, they employ relative minimum confidence term as fitness function to select only the best rules. At the beginning, they propose an algorithm, named ARMGA, which is designed to deal with Boolean association rule mining. However, since they also want to deal with quantitative association rule discovery, they propose another Genetic algorithm based method, named EARMGA which is an expansion of ARMGA. They also designed a FP-tree approach based on FP-growth for implementing EARMGA. Experimental results illustrate that their algorithms reduces computation costs and generates interesting association rules only.

In our previous work [7], we proposed an ARM approach, named ARMICA. Our main focus was on proposing a fast algorithm that extract frequent rules with small time consumption value. Hence, we employed Imperialist Competitive Algorithm (ICA) to extract frequent rules automatically. This approach did not required any predefined minimum support and confidence. Our experimental results illustrated that ARMICA is faster than Apriori. Moreover, ARMICA generates the same rules as Apriori, which can be the proof of its accuracy. However, what was the drawbacks of ARMICA? First, it

requires a predefined parameter, named Number of Imperialists. Although defending a value for this parameter is not a complex task compared to the defining minimum support and confidence, it still relies on user to have this value. Secondly, ARMICA should be compared with other ARM approaches not only the Apriori. Finally, we should consider more parameters to improve the ARMICA, like number of database scans or number of generated rules.

3 Proposed Method

We propose new ARM approach based on ICA algorithm called ARMICA-Improved, which is a heuristic approach. This approach is an improved version of our previous method ARMICA [7]. ARMICA-Improved has some significant improvements compared to the ARMICA. One of the parameter that has a great impact on execution time is the number of database scans. Having higher number of database scans may increase the execution time. For this reason, ARMICA-Improved scans the database only once and at the same time calculate the frequency of each item in the database.

In this algorithm, we consider the frequency of each item as cost of that item in ICA and each item is a country. In addition, we assume that a country with more cost has more power. In the other worlds, a country (item) with high cost (frequency) is powerful. In the first step, the algorithm sends the countries' names and their cost (which were calculated in the database scan stage) to the ICA. ICA selects some of the most powerful countries as imperialists and divides other countries between them based on the power of each imperialist. More powerful imperialist can have more colonies. In this stage, the empires are built. At that point, ICA establishes competition between the empires. The more powerful empires try to steal the colonies of weaker empires. In the original ICA algorithm, the stolen colony become one the new colonies of the powerful empire, but like ARMICA, in ARMICA-Improved, this colony would be removed and added to a list, named Reserve List. Moreover, the stolen colony should be the weakest colony of the weaker empire. This process continues until there is only one colony left for the weaker empire. In this occasion, the colony and imperialist of the weaker empire become colonies of the powerful empire. This completion continues until there is only one empire left. At that point, the algorithm checks if there is any colony in the reserve list, which has more power than any colony in the final empire and exchange them. Finally, the members of the final empire are our frequent itemset. It is noticeable that since ICA selects the most powerful countries as imperialists and because we working on offline databases and the items have fix frequency, there is no chance for a colony to become more powerful than its imperialist; as a result, there is no need for Revolution process.

Next, the algorithm sorts the frequent rules based on their costs and calculates the median cost of them as the universal minimum support. Hence, ARMICA-Improved determines the minimum support automatically and it does not require any user knowledge to set this parameter in advance. One of the differences between ARMICA and ARMICA-Improved is that at this stage ARMICA-Improved removes each items that has frequency less than minimum support. Hence, when it extracts each combination of the remained frequent items, it is rarely to have infrequent itemsets (itemset,

which their support value is less than minimum support). This could reduce the number of generated itemsets, significantly. At that point, the algorithm removes each transaction of the database that does not contain any of the frequent items. It also removes the columns of infrequent items. This process could dramatically decrease the size of the database.

Just like Apriori, FP-growth and ARMICA, ARMICA-Improved tries generate all the possible k-length frequent itemset that k is 1, 2, …, n. However, in contrast to ARMICA, it in each stage, it stores all the generated frequent item sets along with their frequency to avoid any recalculation in the future. This could make the algorithm more efficient compared to the ARMICA. In the other worlds, one of the biggest difference between ARMICA and ARMICA-Improved is that in ARMICA-Improved we calculate the cost of frequent itemsets few times. We store all the costs of all itemsets in previous steps. This save us lots of time compared to ARMICA, which requires calculating costs of all frequent items and itemsets repeatedly.

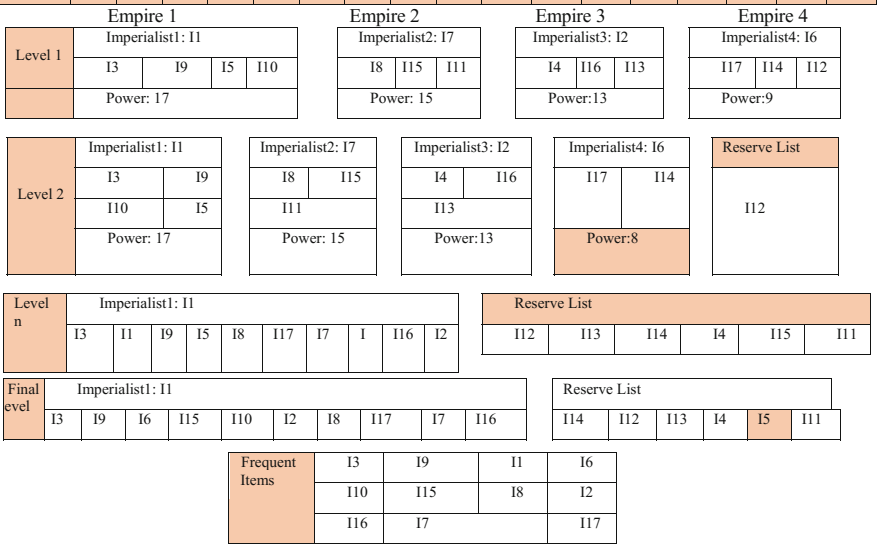
4 Example

To familiarize the readers with ARMICA-Improved, here we made an example. Assume that we have transactional database like Table 1. This database has 17 items and 7 transactions. ARMICA-Improved scans this dataset and calculate the frequency of each items at the same time. At this stage, it sends the items and their frequency to the ICA. ICA selects some of them as imperialist. As it was mentioned before, the number of imperialists is a free parameter in the original ICA. As a result, since here we only have 17 countries, we cannot consider 10 percent of them as imperialist and we assume that we have 4 imperialist and distribute the rest of them between these imperialists based on their power. At this time, we the empires are built. Then the algorithm calculates the power of each empire based on power of their colonies and imperialists. The most powerful empire is empire 1 and the weakest one is empire 4. At this stage. Empire 1 tries to steal the I12 from the empire 4. The algorithm removes this colony from empire 4 and adds it to the reserve list. This process continues until there is only one empire left (Level n). The members of the last empire are the frequent items. ARMICA-Improved stores their names and costs in a list, named Save List for the future references.

Next, the algorithm tries to extract 2-lenth frequent itemsets. It generates them and calculate their costs, and stores them in the Save list. This process continues until all the possible frequent itemsets be produced. Then, ARMICA-Improved extract the frequent rules from these itemsets. Since the algorithms stores all the possible items and temsets and their costs in the Save list, in contrast to the ARMICA, there is no need to calculate the support of different parts of rules again. This would increase the execution time of the algorithm. Finally, ARMICA-Improved generate the frequent rules like other ARM approaches (Table 1).

Table 1. Example database

	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11	I12	I13	I14	I15	I16	I17
T1	t		t		t	t	t		t				t				t
T2	t			t		t		t	t	t	t				t	t	t
T3			t			t	t		t		t				t		
T4	t	t		t			t	t	t			t				t	
T5	t	t					t	t					t		t	t	
T6	t	t	t			t	t			t						t	
T7	t	t		t			t			t				t			T
Frequency	6	4	3	3	1	4	6	3	4	3	2	1	2	1	3	4	3



5 Evaluation

We evaluated ARMICA-Improved 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 2 GB RAM. We also used the implementation of Apriori and FP-growth from Weka 3.6 [8] along with the Supermarket, Mushroom, Spect_Train, and Vote datasets from the UCI Machine Learning Repository [9] to benchmark our method. Moreover, to further study on ARMICA_Improved, we also employed LUCS-KDD ARM data generator [10] to generate different databases with different data density. In the other word, we wanted to investigate our approach under the different circumstances and see what is its characteristics in different databases with different data density. In addition, we considered four factors for this evaluation: the quality of generated rules (formula 2), the number of database scans, the number of generated rules, and execution time. Figure 1 indicates that in the Supermarket dataset ARMICA-Improved has the least number of generated rules with 347 rules. After that, FP-growth generates 350 and Apriori and ARMICA generate 372 rules.

Table 2. The characteristics of databases

Data set	Items	Transac-tions	Input Distribu-tion (Density) %	Algorithm	Predefined Min. Support	Predefined Min. Confidence
Supermarket	217	4627	-	Apriori	28.3	39
				ARMICA	—	—
				FP-growth	28.3	39
				ARMICA-Improved	—	—
Mushroom	119	8124	-	Apriori	36.18	47
				ARMICA	—	—
				FP-growth	36.18	47
				ARMICA-Improved	—	—
DataGenerator 1	110	1200	50	Apriori	1132.3	91
				ARMICA	—	—
				FP-growth	1132.3	91
				ARMICA-Improved	—	—
DataGenerator 2	110	1200	70	Apriori	94	95
				ARMICA	—	—
				FP-growth	94	95
				ARMICA-Improved	—	—
DataGenerator 3	110	1200	40	Apriori	75.8	78
				ARMICA	—	—
				FP-growth	75.8	78
				ARMICA-Improved	—	—
DataGenerator 4	140	2400	55	Apriori	91.43	92
				ARMICA	—	—
				FP-growth	91.43	92
				ARMICA-Improved	—	—
Spect_Train	46	267	-	Apriori	23.14	82
				ARMICA	—	—
				FP-growth	23.14	82
				ARMICA-Improved	—	—
Vote dataset	32	435	-	Apriori	52	86
				ARMICA	—	—
				FP-growth	52	86
				ARMICA-Improved	—	—

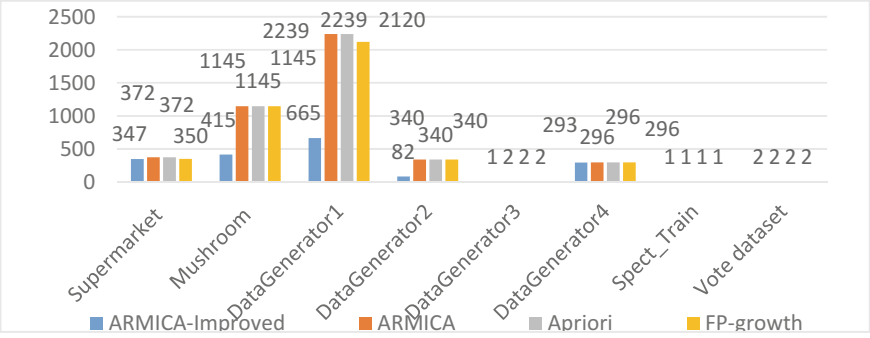


Fig. 1. The number of generated rules

$$\textit{The quality of generated rules} = AVG(\textit{Generated Rules}' \textit{ Confidence}) \tag{2}$$

Figure 2 illustrates that, in the supermarket database the execution time of ARMICA-Improved is the lowest time compared to the other approaches. Its execution time is around 17 times less than Apriori. After ARMICA_Improved, FP-growth, ARMICA have the lowest execution times, respectively. The results in Fig. 3 indicate

that, in the Supermarket database the average quality of generated rules, which we considered it as average confidence value of all generated rules, in ARMICA, Apriori and FP-growth is the same and is equal to 61.09. According to this figure, ARMICA-Improved with 62.2 has the highest average quality of rules compared to the other methods. It is worth to mention that Apriori needs many database scans to generate the frequent rules. After that ARMICA do few scans on the database for mining frequent rules. However, compared to many ARM approaches, FP-growth has one of the lowest number of database scans with 2 scans. This may have effect the execution time of this algorithm. Scanning the database is an I/O operation and having less I/O operation could make the approach faster. ARMICA-Improved with one database scan, scans the database even less than FP-growth.

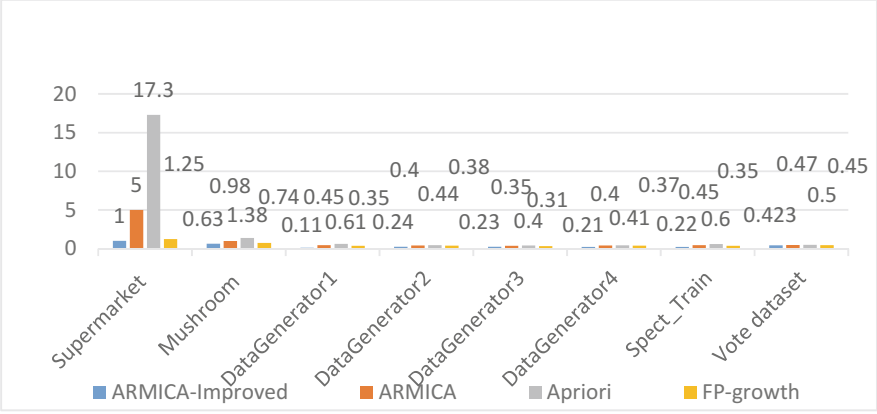


Fig. 2. Execution time

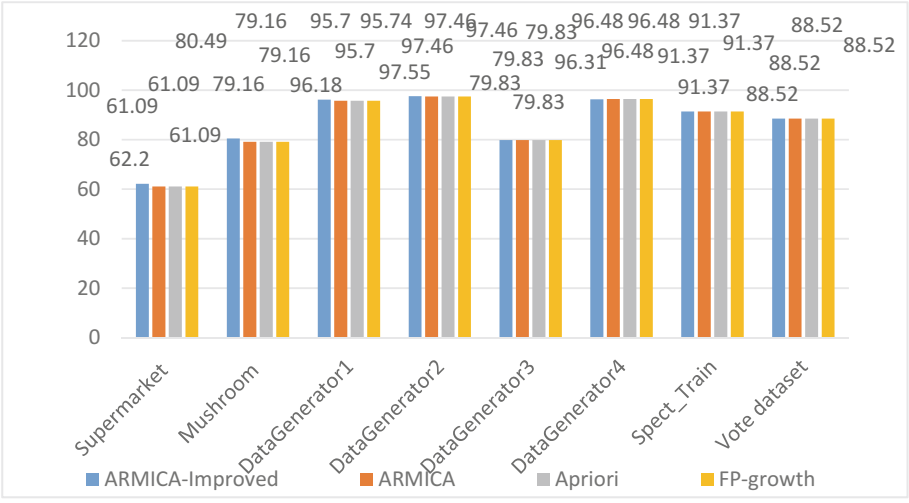


Fig. 3. Average quality of generated rules

6 Discussion

We believe that an ARM approach should optimize more than on parameter at the same time. This makes the approach more productive. At our previous research, ARMICA, we only focused on execution time and automatic procedure. However, in this paper, we considered other parameters like the number of generated rules, the number of database scans, execution time, and the quality of generated rules. Finally, like ARMICA, ARMICA-Improved also is an automatic approach.

Many researches have been done to propose fast ARM approaches. Extracting frequent rules from the database was also one of our primary goal in this paper. Consequently, ARMICA was improved in this paper. ARMICA-Improved tried to decrease the number of database scans, which may have some impact on execution time. It only scans the database once which is even less than number of database scans in FP-growth. Scanning the database is an I/O operation and having less number of I/O operation may decrease the execution time. In addition, ARMICA needs to calculate the frequency of each items and itemsets several times, which could result in increasing the execution time. However, ARMICA-Improved calculates the frequency of items at the same time that it scans the database. Moreover, it has mechanism to decrease the database size. After ICA algorithm finds the frequent items, ARMICA-Improved removes each items (columns) of database that is not frequent. It also removes each transactions, which does not include at least one of the frequent items. This approach will decrease the size of database, significantly. Hence, it is easier to calculate the frequency of each generated itemset in the next steps of the algorithm. In addition, in contrast to ARMICA, ARMICA-Improved stores all the generated itemsets and their frequency in the Save list. This helps the algorithm to prevent any recalculation especially when it tries to calculate the confidence of generated rules; the algorithm easily finds the support of each part of the rules in the save list and find the confidence of those rules.

Although extracting frequent rules in a short period of time is important, the generated rules also should be accurate and have the high quality. In the other words, generating false rules or rules with low quality in a short period of time, may not be suitable for users. As a result, we also considered the accuracy and the quality of extracted rules. Although ARMICA-Improved in the supermarket database produces the least number of rules compared to the Apriori, ARMICA, and FP-growth, it has generated rules with more quality. The experimental results indicate that ARMICA-Improved generates rules with average confidence of 62.2%, which is more than the quality of generated rules by Apriori, ARMICA, and FP-growth with average confidence of 61.09. Moreover, experimental results did not show any false rules generation. Apriori and FP-growth generated all the rules that ARMICA-Improved generated. This could illustrate that the ARMICA-Improved is also an accurate approach.

Finally, like ARMICA, ARMICA-Improved is an automatic approach. Many current ARM approaches require fixed and user defined minimum support and minimum confidence values. Setting these parameters before running the algorithm, especially in the bigger databases, is a hard task. In many case it needs a try and error

approach to set these parameters. As a result, ARMICA-Improved tries to be independent from user knowledge about the database and set the minimum support automatically. This algorithm also does not need any minimum confidence. This feature makes ARMICA-Improved a database independent approach, which could be applied in any databases. However, it requires a predefine parameter like the number of imperialists. Although setting this parameter is not comparable with setting minimum support and confidence, like ARMICA we consider 10 percent of all countries as imperialist, it should be addressed in the future references; we should find a proper mechanism for setting this parameter.

After all, ARMICA-Improved showed some significant improvement in extracting frequent rules from the database. It is a fast approach, scans the database only once, generates less number of rules, generates rules with higher quality, does not generate false rules, removes unnecessary items and transactions from the database, decreases the database's size, and finally, it is an automatic approach and does not need any predefined minimum support and confidence values.

7 Conclusion

With the dramatic increase in amount of available data, applying data mining techniques to extracting useful knowledge from databases became more popular. One of these techniques is ARM. ARM approaches try to extract frequent patterns and rule form the databases. There have been proposed many ARM algorithms; however, many of them only focused on one aspect of the problem. This paper proposed a new ARM approach called ARMICA-Improved, which is an improved version of our previous research, ARMICA. The experimental results indicate that it is faster than Apriori, ARMICA, FP-growth. It scans the database only once; it decreases the size of database; it generates less number of rules and rules with higher quality compared to the other three mentioned approaches. Finally, it is an automatic approach and it does not need any predefined minimum support and confidence. For the future research, we should consider other parameters like interestingness and amount of memory usage. We also should try to test ARMICA-Improved on big data. Finally, ARMICA-Improved needs a proper mechanism to set the number of imperialists, which should be addressed in the future researches.

References

1. Atashpaz-Gargari, E., Lucas, C.: Imperialist competitive algorithm: An algorithm for optimization inspired by imperialistic competition. In: IEEE Congress on Evolutionary Computation, pp. 4661–4667 (2007)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: VLDB Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487–499 (1994)

3. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: ACM SIGMOD Conference on Management of Data. ACM, New York (1993)
4. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Min. Knowl. Disc. **8**, 53–87 (2004)
5. Drias, H.: Genetic algorithm versus memetic algorithm for association rules mining. In: Sixth World Congress on Nature and Biologically Inspired Computing (NaBIC). IEEE, Porto (2014)
6. Yan, X., Zhang, C., Zhang, S.: Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support. Expert Syst. Appl. **36**(2), 3066–3076 (2009)
7. Ghafari, S.M., Tjortjis, C.: Association rules mining using the imperialism competitive algorithm (ARMICA). In: 12th IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI), Thessaloniki (2016)
8. Witten, I.H., Eibe, F., Hall, M.A.: Data Mining: Practical Machine Learning Tools and Techniques, 3rd edn. Morgan Kaufmann, Burlington (2011)
9. Bache, K., Lichman, M.: UCI machine learning repository (2013)
10. Coenen, F.: LUCS-KDD ARM data generator (2007)