High Utility Itemset Mining Using Genetic Approach

Tracy Almeida e Aguiar[1(✉)], Salman Khan[1], and Shankar B. Naik[2]

[1] Rosary College of Commerce and Arts, Navelim, Salcete, Goa, India
tracyaleida@gmail.com

[2] Directorate of Higher Education, Government of Goa, Penha de França, India
xekhar@rediffmail.com

Abstract. Frequent Itemset Mining(FIM) aims to generate itemsets having their frequency of occurrence not lesser than minimum support specified by the user. FIM does not consider the itemset utility which is the it's profit value. High-utility itemset mining(HUIM) mines high-utility itemsets(HUI) from data. HUIM is a combinatorial optimization. With HUIM algorithms, the time required to search increases exponentially with an increasing number of transactions and database items. To address this issue an efficient algorithm to mine HUIs is proposed.

The proposed algorithm uses a compact form of chromosome encoding by eliminating the itemsets with low transactional utilities. The algorithm employs methodology of self mutation to reduce generation of unwanted chromosomes.

Experimental results have shown that the proposed algorithm finds HUIs for a given threshold value. The proposed algorithm consumes less time as compared to another HUIM algorithm HUIM-IGA.

Keywords: High utility itemset mining · Frequent Itemset Mining · Data mining · Genetic Algorithm · Algorithm

1 Introduction

Quick and accurate information is always a necessity to make efficient decisions [2]. The knowledge discovery process(KDD) aims to discover hidden patterns of knowledge from data [1]. One major step of the KDD process is data mining. FIM is an important task in data mining for finding the most occurring itemsets from transactional databases [3]. In FIM itemsets are mined based upon the frequency of occurrence of itemsets in the database only [4,5]. The information about the quantity of the items and profit values associated with the items are not considered [6,7].

HUIM mines itemsets based upon their utilities. Itemset utility is the profit that it offers [8,9]. In this study, the itemset utility is a function of the quantity and profit of the items contained in the itemsets.

The process of HUIM generates a large number of itemsets in the intermediate stages. The itemsets are searched to generate the HUIs. HUIM is a combinatorial optimization problem. The time required to generate HUIs is proportional to the number of items [10,11]. Genetic Algorithms(GA) have the potential to address this issue.

GAs avoid generation of all the candidate itemsets in the intermediate stages. GAs encode itemsets as chromosomes. They generate a set(population) of itemsets, evaluate them for their fitness and perform operations to generate fit itemsets from the weaker ones. The efficiency of GAs in searching new HUIs is low and can be enhanced.

Hence, we propose an efficient version GA to generate HUIs from transactional database, which encodes itemsets as chromosomes where in the genes are identified based upon the utilities of the corresponding item in the database.

2 Related Work and Motivation

2.1 Genetic Algorithm

GAs are used to solve NP-Hard problems [12]. GAs encode itemsets in the form of chromosomes, also called as individuals, which are made up of genes. Each gene represents an itemset. In the initial step GA generates a set of individuals whose fitness(utility) are evaluated to identify the HUIs. In case the required number of individuals are not generated then it performs three types of operations on the generated population. The operations are selection, crossover and mutation. The selection operator selects fit individuals. The crossover operator recombines two of the selected individuals with each other by exchanging their bits of pre-identified genes. The mutation operator alters bits of a single individual to generate a fit individual from it.

2.2 High Utility Itemset Mining(HUIM)

The process of HUIM mines itemsets from transactional datasets which have high utility. The utility is a profit value associated with the it. HUIM was proposed in [14]. The algorithm proposed in [15] is a two phased algorithm which, in the intermediate stages, generates a huge set candidate itemsets. This issue was addressed in [16] which avoided generation of candidate itemsets. The major issue with most of the algorithms in HUIM is huge set of candidate itemsets generated in the intermediate phases. This led to the introduction of GAs in HUIM.

The first GA to mine HUIs was proposed in [10]. Several algorithms thereafter were proposed. The algorithm HUIM-IGA discovered HUIs by using the strategy to improve population diversity [13].

The proposed algorithm stores the HUIs in the form of binary tree which increases the efficiency of searching for an existing HUI. It also maintains bit-vectors of each item which stores information about the ids of the transaction containing the item. This bit-vector enable quick identification of the transactions containing all the items of the itemset and calculation of the itemset utility.

3 Problem Definition

3.1 Preliminaries

Let $I = x_1, x_2, ..., x_m$ be the set of literals, called items. $D = \{T_1, T_2, , ...T_n\}$ represents the database containing n transactions, where T_j represents itemset in transaction j. i is the unique identifier of the transaction and $T_j \subseteq I$.

The external utility of item $x_k \in I$ is the profit value denoted as $\mu(x_k)$. The internal utility denoted as $\nu_j(x_k)$, of item in a transaction is the purchase quantity of the item in that transaction, where $x_k \in I$ is the item in transaction j.

Utility of $x_k \in I$ in T_j is

$$u_{Tj}(x_k) = \mu(x_k) * \nu_j(x_k) \tag{1}$$

Utility $x_k \in I$ in D is

$$u_D(x_k) = \sum_{j=i}^{n} u_{Tj}(x_k), n = |D| \tag{2}$$

Utility T_j in D is

$$u_{Tj} = \sum_{x_k \in T_j} u_{Tj}(x_k) \tag{3}$$

The transactional utility x_k in D is

$$u(x_k) = \sum_{x_k \in T_j} u_{Tj} \tag{4}$$

Utility of X in T_j is

$$u_{Tj}(X) = \sum_{x_k \in T_j \cap X} u_{Tj}(x_k) \tag{5}$$

The utility of itemset X in database D is defined as

$$u(X) = \sum_{X \subseteq T_j} u_{Tj}(X) \tag{6}$$

X is HUI if $u(X) \geq s_0$, where s_0 is minimum utility value given by the user.

3.2 Problem Statement

Generate high utility itemsets for a database D of transactions, given the minimum utility, s_0, and external utilities of items.

4 Proposed Algorithm

The proposed algorithm works in the following steps.

4.1 Database Pruning

The algorithm generates an new database D_{bit} from the transactional database D. The transactions in D_{bit} are bit representations of transactions in D. The columns in D_{bit} represent each item in I. If the item is in a transaction in D then the bit for the corresponding item in the transaction in D_{bit} is set to 1 otherwise is set to 0. The algorithm then identifies the items with their utilities less than s_0. The columns pertaining to these items are deleted from D_{bit}. This reduces the number of comparisons required to generate and search for itemsets in the database. Each column in D_{bit} is a vector of bits containing the transaction ids, in D, to which the item pertaining to the column belongs.

4.2 Initial Population Generation

Each itemset is encoded as a chromosome of length l same as the column count of D_{bit} i.e. the count of items with their utilities not less than s_0.

Let N_p be the size of the population. The algorithm generates the a set of N_p chromosomes in the following way. While generating an individual, all the genes are randomly assigned values 0 or 1. The operator AND is performed between the vectors of columns on each item corresponding to the genes which are set to 1. The position of the 1 valued bits in resultant bit-vector contains the ids of the transaction in D_{bit} containing all the items of the itemset encoded as the new individual (chromosome). The utility of the newly generated itemset is the total of the utilities of all these transactions. If the utility of the itemset is not less than s_0 then the generated individual is added to $HUIS$ as an HUI.

After repeating the process for all the individuals, $HUIS$ contains high utility itemsets.

There are two challenges involved in this step. The first challenge is the generation of duplicate HUIs. In this case, the newly generated itemset will have to be searched for in the existing set $HUIS$. In order to make this search efficient, the set $HUIS$ is maintained as a binary tree of length $l + 2$. The root node is empty. Each branch represents an HUI(individual). The other nodes store bits representing the genes of the individual. When a new HUI is inserted into the tree, the first bit is inserted as a root node child. If gene value is 0 then the node is created as a left child. If value of the gene is 1 then node is created as a right child. If the corresponding child node already existed then nothing is done. If the value of the gene is 0 then the algorithm considers the left child node for the second gene node creation. Otherwise the right child node is considered.

Similarly, for the second gene, the child node is created at level 3 in the binary tree base on the value of the second gene in the same way as done for the first gene. The process is repeated till all the genes are processed and a new branch of nodes of depth $l+1$ is added to the tree. If a new branch is not created and there already existed a branch for the itemset then it implies that the HUI has already been created before. Thus there is no extra step required to check whether an HUI has been already created. This tree structure also avoids comparison with individual HUIs in the set $HUIS$ while searching for an itemset.

The second challenge is that not all the individuals generated during this process will qualify to be high utility itemset. If this happens then the number of HUIs in the set $HUIS$ will be less than N_p.

In both the cases, i.e. if the generated itemset is already existing in set $HUIS$ or the itemset is not an HUI, then itemset is made to undergo mutation and a new itemset is created by exchanging values of randomly selected two genes such that both the genes have different values. The new itemset is evaluated for its fitness and checked whether it has been already included before in set $HUIS$. If not, then it also undergoes mutation.

4.3 Time Complexity

The algorithm has the time complexity of $N_p * l + N_p^2 * l$, where N_p is the size of population and l is the length of the chromosome.

5 Experiments

The performance of the proposed algorithm was compared with HUIM-IGA. The algorithm proposed in this paper was implemented using C++. The experiments were conducted on a 64-bit Intel Core-i5 processor system having 8 GB RAM and Windows 10 operating system.

The dataset used is a synthetic dataset which as generated using the IBM synthetic generator. Size of the population is 20. The number of fitness calculations is 30K. The value of s_0 was set to 40%.

Figure 1 shows the convergence of both the proposed and HUI-IGA algorithms with respect to the count of HUIs generated.

The convergence of the proposed algorithm is faster than the convergence of HUIM-IGA. The number of HUIs with lesser fitness evaluation calculations in the proposed algorithm are more as compared to that in HUIM-IGA.This is due to the intersections of the item bit-vectors at the time of generation of a new individual.

Figure 2 shows the execution time of both, the proposed and HUI-IGA algorithms with respect to the minimum utility threshold s_0.

For lower values of s_0 the proposed algorithm requires more time than HUIM-IGA. The proposed algorithm is better in terms of time efficiency as compared to the HUIM-IGA for higher values of s_0. Since the proposed algorithm stores the HUIs in the form of a tree which avoid unnecessary comparisons required for searching HUIs, the execution time is lower than that of HUIM-IGA.

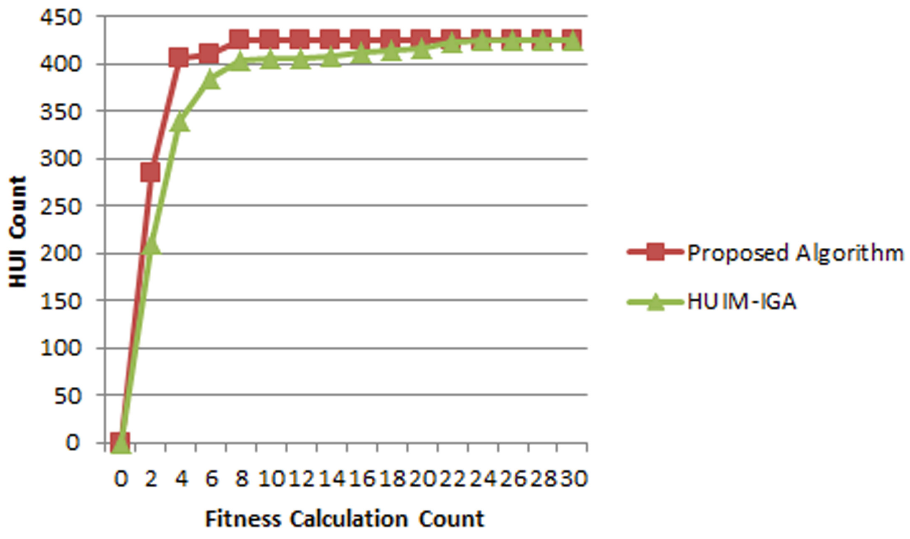

Fig. 1. Convergence with $s_0 = 40\%$

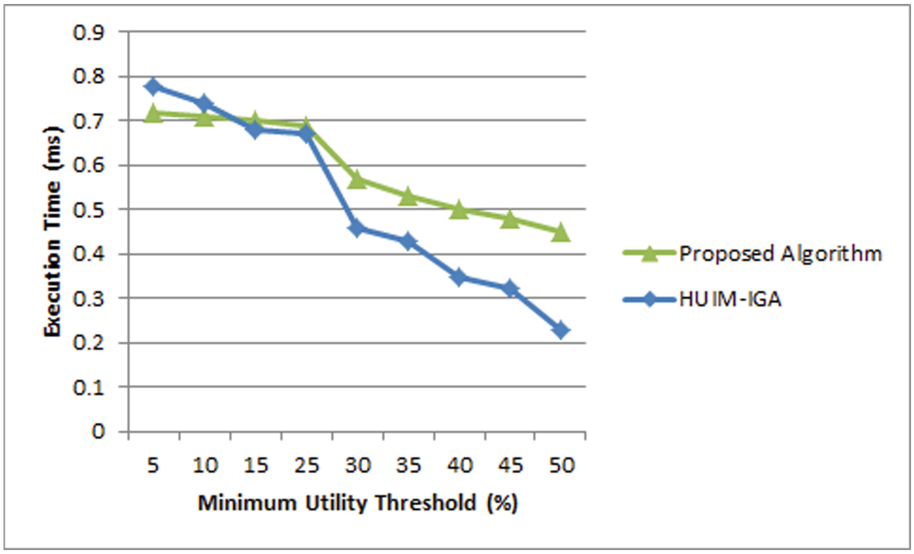

Fig. 2. Execution time vs s_0

6 Limitation of the Study

The proposed algorithm has been experimented on one and only synthetic data. Experiments on multiple dataset and real datasets will enhance the experimental study.

There are possibilities that the proposed algorithm may enter infinite loop while generating a new individual out of duplicate HUI or an unfit individual in the initial population generation stage. A control strategy to avoid repetitive generating of new individuals forever is needed.

Only one algorithm has been considered for comparison. The performance has not been compared with other algorithms. Also, the observations are true for the current dataset used. Whether the same trends will or will not be followed for other datasets has to analysed.

7 Conclusion

HUIM aims to discover itemsets having high utility. HUIM algorithms generate large no of itemsets out of which the HUIs are discovered. This issue has been addressed by Genetic Algorithms. GAs generate a set of individuals and evaluates their fitness. In case of an unfit individual, operations such as mutations and crossover are performed to either convert a weak individual into a fit one or generate a new individual from two parents. In case of high utility itemset mining the individuals represent the itemsets. GAs also have to maintain and search large number of itemsets in their intermediate steps.

A GA which stores HUIs in the form of a tree has been proposed in this paper. This reduces the unnecessary comparisons thereby improving the search efficiency and reducing the overall execution time. The algorithms also maintains the bit image of the transactional database which enables quick calculations of utilities of itemset by performing AND operations between the bi-vectors of items.

The proposed algorithm was implemented in C++ to perform experiments to compare its performance with the state-of-art GA algorithm HUIM-IGA. As per the experiments the proposed algorithm is efficient in mining HUIs for high s_0 values.

A limitation of the proposed algorithm is that the process of generating a new HUI from a duplicate HUI or low utility itemset may enter an infinite loop. The strategy to prevent it from entering an infinite loop is required. Only synthetic data were used in the experiment. A better insight about the performance of the algorithm would be possible using real datasets. Our future work will focus on these issues.

References

1. Han, J., Kamber, M., Pei, J.: Data Mining: Concepts and Techniques, 3rd edn. Morgan Kauffman, Burlington (2011)

2. Barretto, H.M., Dessai, P.S.: Challenges faced by Academic Libraries due to resource sharing and networking models. Libr. Philos. Pract. 1–14 (2021)
3. Zaki, M.J., Meira, W., Jr., Meira, W.: Data Mining and Analysis: Fundamental Concepts and Algorithms. Cambridge University Press, Cambridge (2014)
4. Naik, S.B., Pawar, J.D.: An efficient incremental algorithm to mine closed frequent itemsets over data streams. In: Proceedings of the 19th International Conference on Management of Data, pp. 117–120, December 2013
5. Naik, S.B., Pawar, J.D.: A quick algorithm for incremental mining closed frequent itemsets over data streams. In: Proceedings of the Second ACM IKDD Conference on Data Sciences, pp. 126–127, March 2015
6. Naik, S.B., Khan, S.: Application of Association Rule Mining-Based Attribute Value Generation in Music Composition. In: Bhateja, V., Satapathy, S.C., Travieso-González, C.M., Aradhya, V.N.M. (eds.) Data Engineering and Intelligent Computing. AISC, vol. 1407, pp. 381–386. Springer, Singapore (2021). https://doi.org/10.1007/978-981-16-0171-2_36
7. Amballoor, R.G., Naik, S.B.: Utility-based frequent itemsets in data streams using sliding window. In: 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), pp. 108–112. IEEE, February 2021
8. Chan, R., Yang, Q., Shen, Y.D.: Mining high utility itemsets. In: Third IEEE International Conference on Data Mining, pp. 19–19. IEEE Computer Society, November 2003
9. Lin, J.C.W., et al.: Mining high-utility itemsets based on particle swarm optimization. Eng. Appl. Artif. Intell. **55**, 320–330 (2016)
10. Kannimuthu, S., Premalatha, K.: Discovery of high utility itemsets using genetic algorithm with ranked mutation. Appl. Artif. Intell. **28**(4), 337–359 (2014)
11. Pattern Mining with Evolutionary Algorithms. Advances in Intelligent Systems and Computing, Springer, Cham (2016). https://doi.org/10.1007/978-3-319-33858-3
12. Karakatič, S., Podgorelec, V.: A survey of genetic algorithms for solving multi depot vehicle routing problem. Appl. Soft Comput. **27**, 519–532 (2015)
13. Zhang, Q., Fang, W., Sun, J., Wang, Q.: Improved genetic algorithm for high-utility itemset mining. IEEE Access **7**, 176799–176813 (2019)
14. Yao, H., Hamilton, H.J., Butz, C.J.: A foundational approach to mining itemset utilities from databases. In: Proceedings of the 2004 SIAM International Conference on Data Mining, pp. 482–486. Society for Industrial and Applied Mathematics, April 2004
15. Liu, Y., Liao, W., Choudhary, A.: A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005). https://doi.org/10.1007/11430919_79
16. Li, Y.C., Yeh, J.S., Chang, C.C.: Isolated items discarding strategy for discovering high utility itemsets. Data & Knowledge Engineering **64**(1), 198–217 (2008)