Expert Systems with Applications 38 (2011) 288–298

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Multi objective association rule mining with genetic algorithm without specifying minimum support and minimum confidence

Hamid Reza Qodmanan *, Mahdi Nasiri, Behrouz Minaei-Bidgoli

Department of Computer Engineering, Iran University of Science and Technology, Tehran, Iran

ARTICLE INFO

Keywords:
Data mining
Association rule mining
Genetic algorithm
Multi objective

ABSTRACT

Multi objective processing can be leveraged for mining the association rules. This paper discusses the application of multi objective genetic algorithm to association rule mining. We focus our attention especially on association rule mining. This paper proposes a method based on genetic algorithm without taking the minimum support and confidence into account. In order to improve algorithm efficiency, we apply the FP-tree algorithm. Our method extracts the best rules that have best correlation between support and confidence. The operators of our method are flexible for changing the fitness. Unlike the Apriori-based algorithm, it does not depend on support. Experimental study shows that our technique outperforms the traditional methods.

1. Introduction

Data mining is the most instrumental tool in discovering knowledge from transactions (Chen, Hong, & Tseng, 2006; Kayaa & Alhajj, 2005; Tsay & Chiang, 2005). Nowadays, improvement in technology allows stores to collect several data about customer's market baskets. A basket indicates items purchased by a customer at a specific time. Customer's purchases can be prospected by analyzing market baskets. It helps vendors for future schematizations. The most important application of data mining is discovering association rules. This is one of the most important methods for pattern recognition in unsupervised systems. This data mining method is very similar to people searching for gold in a very large data base. Here, gold is an interesting rule which has not been discovered yet. These methods find all possible rules in the database. However, it can be considered as a disadvantage because the large number of discovered rules makes it difficult to analyze them. Some measures like support and confidence are used to indicate high quality rules. Most of the association rule algorithms are based on methods proposed by Agrawal, Imielinski, and Swami (1993) and Agrawal and Srikant (1994), Apriori (Agrawal et al., 1993), SETM (Houtsma & Swami, 1993), AIS (Agrawal et al., 1993), Pincer search (Lin & Kedem, 1998) etc. Neither the rules with numeric attribute nor the rules in the form of $I_8I_{10}I_{12} \rightarrow I_4I_5I_9$ can be discovered by these methods. We must appointment minimum support and minimum confidence in previous method. This particular makes these methods depended on datasets and it must execute several time. We do not require to appointment support and confidence in our method and extract best rules in once executed. Previous method mine association rule in two stages. First they find frequent itemsets and then extract association rules from frequent itemsets. This paper proposed a method based on genetic algorithm without considering minimum support, confidence and interestingness, and extract best rule with high of them. If we do not require to use support, we can not use apriori base method for rule mining, whereas in our method we can unuse support with change fitness. Our method is very flexible on changing fitness, so user can define any normal multi objective fitness with support, confidence and etc. and obtains his interesting rules. Moreover, he can define the fitness function so that the order of items is considered on importance of rules.

The rest of this paper is organized as follows. In Section 2 we state the preliminaries of our method, in Section 3 we present our method, in Section 4 we experimentally evaluate our approach. Finally, we conclude our work in Section 5.

2. Preliminaries

In this section, we state the preliminaries of our method. First, we explain the association rule and genetic algorithm. Afterward, we define some measures which rules are evaluated by them and, finally, we state some related works on this subject.

2.1. Association rule

$I = \{i_1, i_2, \ldots, i_m\}$ is a set of literals, or items. X is an itemset if it is a subset of I. Itemset X is a k-itemset if X exactly has k items.

* Corresponding author.
E-mail addresses: ghodmanan@comp.iust.ac.ir (H.R. Qodmanan), nasiri@comp.iust.ac.ir (M. Nasiri), b_minaei@iust.ac.ir (B. Minaei-Bidgoli).

0957-4174/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2010.06.060

$D = \{t_1, t_2, \ldots, t_n\}$ is a set of transactions, called the transaction database, where each transaction t_i has a transaction identifier $t_i d_i$, and a k_i-itemset X_i, that is, $t_i = (t_i d_i; X_i),\ 1 \leqslant k_i \leqslant m;\ i = 1, \ldots, n$. A transaction, t_i, contains an itemset, X, if and only if, for any item $i \in X$, i is in t-itemset X_i.

There is a natural lattice structure, namely the subset/superset structure, over the set of all itemsets, 2^I. Some certain sub-lattice of it can be taken as taxonomy.

An itemset, X, in a transaction database, D, has a support, denoted as sup(X) or simply $p(X)$, that is the ratio of transactions in D containing X. Or

$$\sup(X) = |X(t)|/|D|,$$

where $X(t) = \{t \text{ in } D | t \text{ contains } X\}$, and $|X(t)|$ and $|D|$ are the numbers of transactions in $X(t)$ and D, respectively.

An itemset, X, in a transaction database, D, is called a large (frequent) itemset if its support is equal to, or greater than, a threshold of minimal support (minsupp), which is given by user or expert.

An association rule is an implication $X \to Y$, where itemsets X and Y do not intersect.

Each association rule has two quality measurements, support and confidence, defined as follows:

- the support of a rule $X \to Y$ is the support of $X \cup Y$, where $X \cup Y$ is the union of X and Y;
- the confidence of a rule $X \to Y$, written as $conf(X \to Y)$, is the ratio $|(X \cup Y)(t)|/|(X(t))|$ or $\sup(X \cup Y)/\sup(X)$.

That is, support implies frequency of occurring patterns, and confidence means the strength of implication. We now introduce the *support-confidence framework* (Agrawal et al., 1993).

Let I be the set of items in database D, $X, Y \subseteq I$ be itemsets, $X \cap Y = \emptyset$; $p(X) \neq 0$ and $p(Y) \neq 0$. The minimal support, *minsup*, and the minimal confidence, *minconf*, are given by user or expert. Then rule $X \to Y$ is valid if

$$\sup(X \cup Y) \geqslant minsup, \quad \text{and}$$
$$conf(X \to Y) \geqslant minconf.$$

Mining association rules can be taken into the following two subproblems.

(1) Generating all itemsets for which supports are greater than, or equal to, the user-specified minimum support, that is, generating all large itemsets; and
(2) Generating all the rules which satisfy the minimum confidence constraint in a naive way as follows. For each large itemset X, and any $B \subset X$, let $A = X - B$. If the confidence of a rule $A \to B$ is greater than, or equal to, the minimum confidence (or $\sup(X)/\sup(A) \geqslant minconf$), then $A \to B$ can be extracted as a valid rule (Yan, Zhang, & Zhang, 2008).

2.2. Genetic algorithm

Genetic algorithms are implemented as a computer simulation in which a population of abstract representations (called chromosomes or the genotype of the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0 and 1 s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. This stage is defined Initialization. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. If the number of chromosome in new population is larger than the specified value, a certain number of them select as next population and is used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

Typical genetic algorithm requirements:

- a genetic representation of the solution domain,
- a fitness function to evaluate the solution domain.

A standard representation of the solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, that facilitates simple crossover operation. Variable length representations may also be used, but crossover implementation is more complex in this case.

The fitness function is defined over the genetic representation and measures the *quality* of the represented solution. The fitness function is always problem dependent. For instance, in the knapsack problem, one wants to maximize the total value of objects that can be put in a knapsack of some fixed capacity. A representation of a solution might be an array of bits, where each bit represents a different object, and the value of the bit (0 or 1) represents whether or not the object is in the knapsack. Not every such representation is valid, as the size of objects may exceed the capacity of the knapsack. The *fitness* of the solution is the sum of values of all objects in the knapsack if the representation is valid, or 0 otherwise. In some problems, it is hard or even impossible to define the fitness expression; in these cases, interactive genetic algorithms are used.

Once we have the genetic representation and the fitness function defined, GA proceeds to initialize a population of solutions randomly, and then improve it through repetitive application of mutation, crossover, inversion and selection operators.

2.3. Multi objective

Some of researches tried to visualize association rule mining as a multi objective problem rather than single objective one (Davis, 1991).

Several measures can be considered as objective. Having the rule: $X \to Y$ where both X and Y are candidate itemsets; here is the list of most common objectives:

Support count of the rule is the number of records containing both X and Y itemsets.

Confidence factor or predictive accuracy of a rule is defined as:

$$\text{Confidence} = \frac{\text{Support}(X \cup Y)}{\text{Support}(X)}.$$

Comprehensibility of an association rule is quantified by the following expression:

$$\text{Comprehensibility} = \frac{\log(1 + |Y|)}{\log(1 + |X \cup Y|)},$$

where |itemset| means the number of attributes involved in the itemset. As a simple sentence, if the number of conditions in the antecedent part is less, the rule is more comprehensible.

Interestingness measure: is used to quantify how much the rule is surprising for the user. As the most important purpose of association rule mining is to find some hidden information, it should extract

rules that have comparatively less occurrence in the database. The following expression can be used to quantify the interestingness:

$$\text{Interestingness} = \frac{\text{Support}(X \cup Y)}{\text{Support}(X)} \times \frac{\text{Support}(X \cup Y)}{\text{Support}(Y)} \times \left(1 - \frac{\text{Support}(X \cup Y)}{\text{Support}(Z)}\right),$$

where Support(Z) is the number of records in the database (Davis, 1991).

We can use any of above measures in our fitness function for evaluating of rules.

2.4. Related works

In the early years, some optimization methods for association rule mining have been proposed. The process is too resource consuming, especially when there is not enough available physical memory for the whole database. A solution to encounter this problem is to use genetic algorithm, which reduces both cost and time of rule discovery. Genetic algorithm, colony algorithm, evolutionary algorithm and particle swarm algorithm are instances of single objective association rule mining algorithms. A few of these algorithms has been used for multi objectives. Kaya proposed genetic clustering method (Kayaa & Alhajj, 2005). Hong proposed a cluster-based method for mining generalized fuzzy association rules (Chiu, Tang, & Hsieh, 2008). Ghun proposed a cluster-based fuzzy-genetic mining method for association rules and membership functions (Chen et al., 2006). Dehuri et al. proposed a rule mining method using multi objectives called multi objective genetic algorithm (MOGA) (Dehuri & Ghosh, 2004). Later, they improved the performance by parallel association rule (Dehuri & Ghosh, 2004). Nasiri et al. optimized the previous method using clustering which causes the database to be scanned just once at all (Hadian, Nasiri, & Minaei-Bidgoli, 2009). They have proposed a method for rule mining with PSO (Esmaeli, Nasiri, Minaei-Bidgoli, & Mozayani, 2008) and numeric rule mining with simulated annealing (Nasiri, Tagavi, & Minaei-Bidgoli, 2010). Alatas et al. proposed a multi objective differential evolution algorithm for mining numeric association rules (Alatas & Akin, 2007). Later, they proposed another numeric association rule mining method using rough particle swarm algorithm which had some improvements in performance and precision compared to the previous one (Alatas & Akin, 2008a). They also proposed another numeric association rule mining method chaos rough particle swarm algorithm (Alatas & Akin, 2008b). Yan proposed a method based on genetic algorithm without considering minimum support (Yan et al., 2008). The method uses an extension of elaborate encoding while relative confidence is the fitness function. A public search is performed based on genetic algorithm. As the method does not use minimum support, a system automation procedure is used instead. It can be extended for quantitative-valued association rule mining. In order to improve algorithm's efficiency, it uses a generalized FP-tree. Evaluation of the algorithm shows a considerable reduction in cost of calculation. Just interesting rules with constant length are discovered. In this method, the genes contain rank of fields. Final chromosome should be the best one and the process stops if it reaches the predefined number of iterations or the result is not improved. The fitness function is defined such a way that it causes many rules to be generated and the initialization and crossover operator are defined such a way that the algorithm stays in local optimum and the crossover generates invalid rules. They evaluated rules only with single objective.

3. Identifying association rule with genetic algorithms

In this section, we explain our method. First, we define the fitness function and encode the problem. Then we define the genetic algorithm operators and initialization function and finally, explain the ARMMGA algorithm.

3.1. Fitness function

The definition of ARMGA fitness cause that it returns many rules. Therefore, we eliminate this weakness with defining a new fitness function. Also, we want to find the association rules that both their support and confidence are larger than the other rules. So we define the fitness function as shown in (1)

$$\frac{(1 + \sup(X \cup Y))^2}{1 + \sup(X)}. \tag{1}$$

In this equation, $\sup(X \cup Y)$ is the support of $X \to Y$ and $\sup(X)$ is the support of antecedent part of it. The confidence of rule is $\frac{\sup(X \cup Y)}{\sup p(X)}$. For influence of support and confidence in finding rules, difference fitness functions are suggested. In our suggested method, we multiply those together, so the result is shown in (2)

$$\frac{\sup(X \cup Y)^2}{\sup(X)}. \tag{2}$$

From definition of support, we know:

$$0 \leqslant \sup(X \cup Y) \leqslant 1, \tag{3}$$

$$\sup(X \cup Y) \leqslant \sup(X). \tag{4}$$

We obtain (5) from (2):

$$\sup(X \cup Y)^2 \leqslant \sup(X \cup Y). \tag{5}$$

So we can result (6) from (3)–(5):

$$0 \leqslant \frac{\sup(X \cup Y)^2}{\sup(X)} \leqslant 1. \tag{6}$$

Now, for more discrimination between rules, both $\sup(X \cup Y)$ and $\sup(X)$ sum with one. So the result becomes

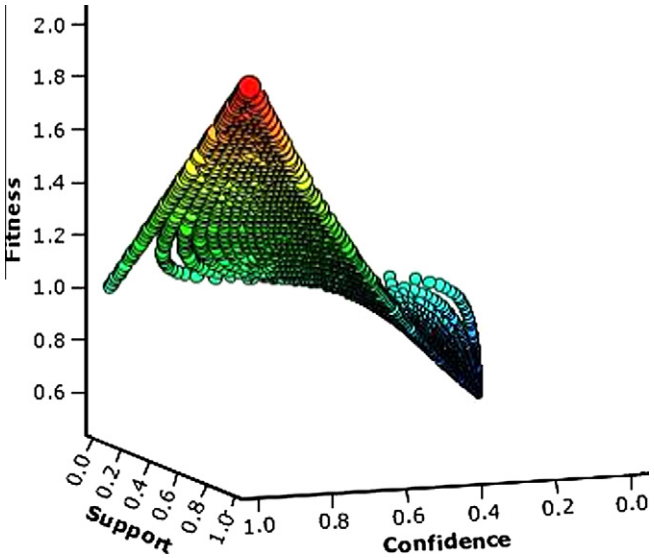


Fig. 1. The fitness function values according to support and confidence.

j	A_1	...	A_j	A_{j+1}	...	A_k

Fig. 2. Encoding of a k-rule.

```
population select (pop,sp)
begin
    selected-pop ← ⊘;
    sort (pop);
    for ∀ chr ∈ pop do
     begin
       if (frand() * (size (pop) – numchr(pop,chr))/ size (pop) ≥ 1-sp)
          selected-pop ← selected-pop ∪ { chr };
     end
     retrun selected-pop;
end
```

Fig. 3. The semi code of selection function.

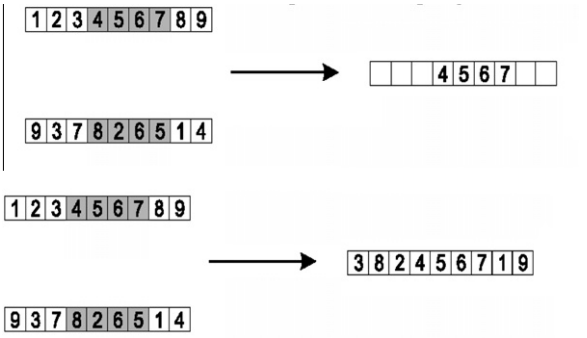


Fig. 4. Order-1 crossover.

```
population crossover (pop,cp)
begin
    pop-child ← ⊘;
    sort (pop);
    for ∀ chr1, chr2 ∈ pop and chr1 ≠ chr2 do
    begin
          if (frand() * (size (pop) – numchr(pop,chr1))/ size (pop)
             * (size (pop) – numchr(pop,chr2))/ size (pop) ≥ 1-cp)
          begin
             child1,child2 ← orderCrossover (chr1,chr2);
             pop-child ← pop-child ∪ {child1,child2};
          end
    end
     retrun pop-child;
end
```

Fig. 5. Semi code of crossover operator.

```
population mutate (pop , cp)
begin
    sort (pop);
    for ∀ chr ∈ pop do
     begin
          if (frand() * (size (pop) – numchr(pop,chr))/ size (pop) ≥ 1-mp)
          begin
              listitems[] ← itemsnotchr (chr);
              newsep ← frand (1,k-1);
              numgene ← frand (1,k);
              chr[0] ← newsep;
              chr[numgene] ← listitems[frand (1,size(listitems))];
          end
     end
     retrun pop;
end
```

Fig. 6. The semi code of mutation.

$$0.5 \leqslant \frac{(1+\sup(X \cup Y))^2}{1+\sup(X)} \leqslant 2. \tag{7}$$

We use Eq. (7) as fitness function in our method. This function shows different values of corresponded values of support and confidence. Increasing the support or confidence increases the value of this function, this is shown in Fig. 1.

```
population selectnextpop (parents-pop,childs-pop)
begin
    next-pop ← ⊘;
    next-pop ← addBestChild(childs-pop,3);
    while size (next-pop) < maxpopsize do
    begin
        parent-random-pop ← ⊘;
        child-random-pop ← ⊘;
        best-chr ← ⊘;
        parent-random-pop ← selectrandomchr (parents-pop,5);
        child-random-pop ← selectrandomchr (childs-pop,5);
        best-chr ← get&removebestchr (parent-random- pop, child-random-pop);
        next-pop ← next -pop ∪ { chr };
    end
    return next-pop;
end
```

Fig. 7. The semi code of final population selection operator.

```
population intialize ()
begin
    int-pop ← ⊘;
    listItem ← createListOfAllItem();
    while size (int-pop) < maxpopsize do
    begin
      chr.A0 ← frond(1,k-1);
      if ( k ≤ size (listAtt) )
        begin
        while size (chr) < k+1 do
          begin
            newItem ← listItem [frand ( size (listItem) )];
            addGene (chr , newItem);
            remove (newItem , newItem);
          end
        end // end if
else
      begin
      for ∀ item ∈ listItem do
          begin
          addGene (chr , item);
        end // end for
      noExistList ← createNoExistItem ( listItem);
      while size (chr) < k+1 do
        begin
          newItem ← noExistItem [frand ( size (noExistItem) )];
          addGene (chr , newItem);
          remove (noExistItem, newItem);
        end
      listItem ← listItem ∪ noExistItem;
      end // end else
    int-pop ← int-pop ∪ { chr };
    end // end while
    return int-pop;
end
```

Fig. 8. The semi code of initialization.

A problem that may be occurred by the this function is that if an initial population includes small fitness chromosome, the genetic algorithm operators filter major chromosomes in the early states and it converges in several early states, so optimal rules are not found. Also, major rules that have high confidence, their support usually is less than 0.5, so the utmost value of this fitness function is usually closely 60% of its maximized value. In three defined operators (selection, crossover and mutation), early, the fitness value is normalized between 0 and 1, and then other acts are done. So, many chromosomes may be filtered in early states. To prevent this problem, we can sort population decently according to its chromosomes fitness and then assign $\frac{n}{n}\cdots\frac{1}{n}$ values to each chromosome fitness, so that the best chromosome in the population gets $\frac{n}{n}$ and the worse chromosome gets $\frac{1}{n}$. Therefore, by this change, we can make a good and sufficient press for better chromosomes and if the fitness values are low or their difference is high in the initial population, we can prevent from hasty converge.

```
population ARMMGA (sp,cp,mp)
begin
  i ← 0;
  pop[i] ← intialize ();
  best-pop ← pop[i];
  while not terminate (pop[i]) do
  begin
     parents-pop ← ⊘;
     childs-pop ← ⊘;
     parents-pop ← select (pop[i],sp);
     childs-pop ← crossover (parents-pop,cp);
     childs-pop ← mutate (childs-pop , cp);
     pop[i+1] ← selectnextpop (parents-pop,childs-pop);
     if (means (best-pop) < means (pop[i+1]) or
        ( means (best-pop) = means (pop[i+1]) and
         variance (best-pop) ≥ variance (pop[i+1])) )
     begin
        best-pop ← pop[i+1];
     end
     i ← i+1;
  end
  return best-pop;
end
```

Fig. 9. The semi code of ARMMGA.

Table 1
The specifications of datasets.

	Balance scale	Monk's problems	Nursery	Solar Flare	SPECT heart
N. record	625	431	12,960	1066	267
N. attribute	23	19	32	48	46

Table 2
The parameters for run ARMMGA.

	SPECT heart	Solar Flare	Nursery	Monk's problems	Balance scale
sp	0.95	0.95	0.95	0.95	0.95
mp	0.85	0.85	0.85	0.85	0.85
cp	0.01	0.01	0.03	0.03	0.01
db	0.05	0.05	0.01	0.01	0.01
max si	25	25	25	25	25
min si	6	6	6	6	6
mi	25	25	25	25	25

We now state our task of association rule mining as follows:

Problem 1: Given a rule length k, we search for some high-quality association k-rules, with their fitness function acceptably maximized, by using a genetic algorithm.

3.2. Encoding

After defining of fitness function, we encode our problem to solve it by genetic algorithm. This encoding is get from paper (Yan et al., 2008).

Table 3
The parameters for running the ARMGA.

	SPECT heart	Solar Flare	Nursery	Monk's problems	Balance scale
sp	0.95	0.95	0.95	0.95	0.95
mp	0.85	0.85	0.85	0.85	0.85
cp	0.01	0.01	0.01	0.01	0.01
db	0.01	0.01	0.01	0.01	0.01
si	25	25	25	25	25
mi	25	25	25	25	25

Table 4
The obtained rules of two algorithms.

	Balance scale	Monk's problems	Nursery	Solar Flare	SPECT heart
N. rule ARMMGA	34	4	4	23	23
N. rule ARMGA	25	36	62	313	67

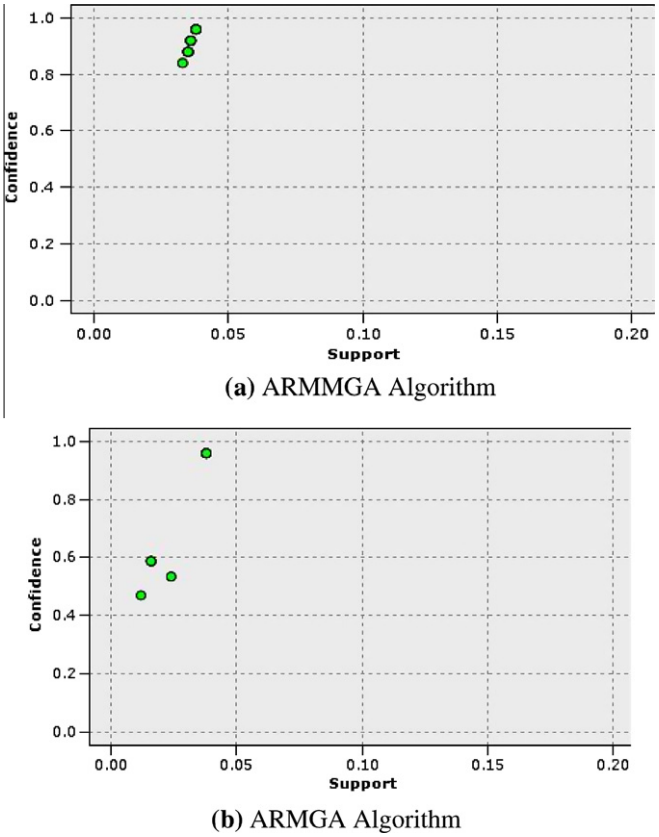


Fig. 10. The variance of obtained rules from balance scale dataset.

First, we assign an index to each item so that the index of item i is i. Then encode $X \rightarrow Y$ rule according to Fig. 2 where j is an indicator that separates the antecedent from the consequent of the rule. That is, $X = \{A_1, \ldots, A_j\}$ and $Y = \{A_{j+1}, \ldots, A_k\}$; $0 < j < k$. Therefore, a k-rule $X \rightarrow Y$ is represented by $k + 1$ positive integers.

3.3. Parent selection

The selection is one of genetics algorithm operator that was defined in Section 2.2. Our parent select operator is get from paper (Yan et al., 2008). The *select* (pop,sp) acts ac filter of chromosomes with considerations of their fitness and selection probability.

The semi code of this function is showed in Fig. 3.

frand () function return a random real number ranged from 0 to 1. *sort* (pop) function sorts the population decently and *numchr* (pop,chr) function returns the chromosome number in population.

3.4. Crossover operator

It is the other genetic algorithm operator that was explained it in Section 2.2. In paper (Yan et al., 2008), the two-point strategy is used for this operator. The disadvantage of the strategy is that invalid chromosome may be produced so that in one chromosome sum genes has same value and testing of chromosome validation is time consuming. Therefore for preventing of them, we use the Order-1 crossover. In this strategy, in first step, one segment is selected equally from two chromosomes and, respectively, are copied from first and second parents to first and second offspring and in second step, then, we start from right side of segment and copy the value of genes that no exist in the selected segment of first parent, to first offspring. We do this act similarly for second offspring (Eiben & Smith, 2003) for more illustration see Fig. 4.

The semi code of crossover operator is showed in Fig. 5. In this operator, all chromosomes that select from *select* function, marriage together and the reproduced offspring return at new population. cp is crossover probability and *frand* () function returns a random real number from 0 to 1. The *orderCrossover* (chr1,chr2) do the order-1 crossover on two chromosomes and return the reproduced offspring.

3.5. Mutation operator

This operator is explained in Section 2.2. In suggested algorithm for it, with use of methods that come in Eiben and Smith (2003), such as swapping and ..., we can only change the confidence of rules. So we can do this action with change the separator value (first gene) simply and sufficiently. For creating of new support, we only change the value of one of genes randomly but it may produce invalid chromosome. So, to prevent from producing of invalid chromosome, we produce a list of all number items and delete all items that exist in the chromosome and randomly select an item that remains in the list and finally substitute the selected value with a value of gene in chromosome. The semi code of this operator is showed in Fig. 6. In it, mp is mutation probability, *itemnotchr* (chr) returns a list of number items that no exist in the chromosome chr, the *frand* $(1, k)$ returns a random integer ranged from 1 to k. In this semi code, first, a random integer number from 1 to $k - 1$ that separates the antecedent from consequence is produced

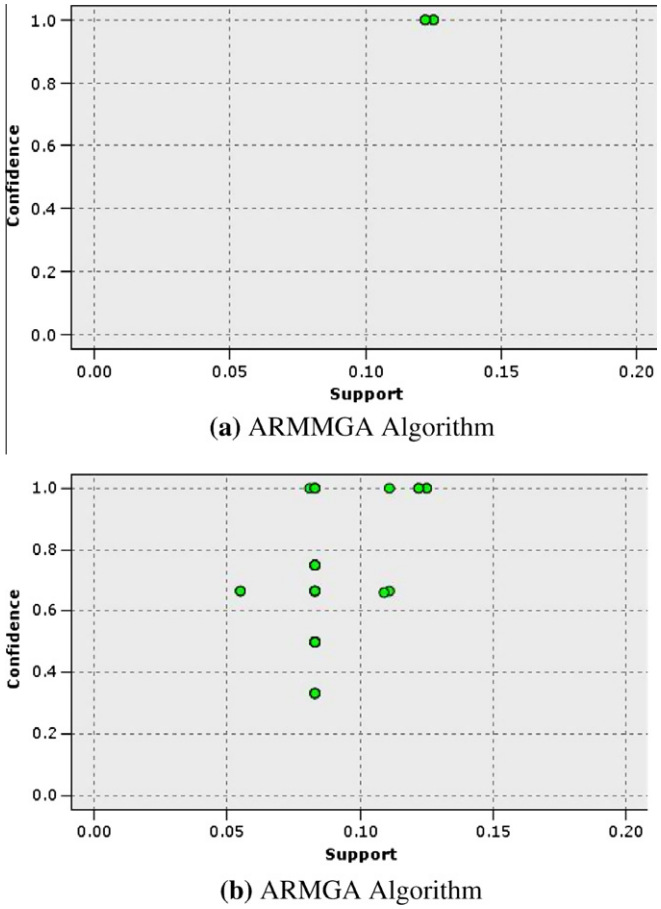


Fig. 11. The variance of obtained rules from Monk's problems dataset.

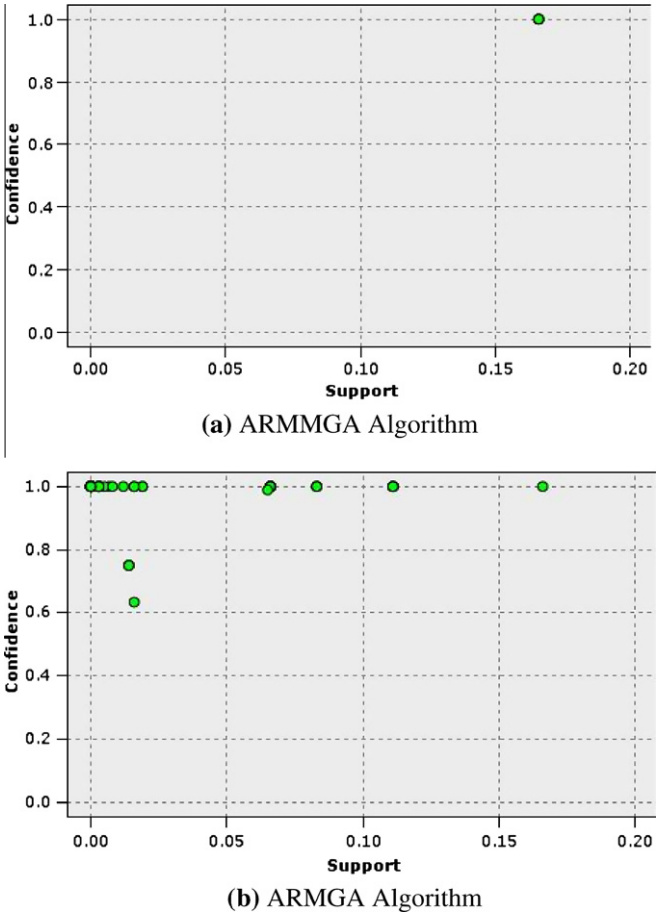


Fig. 12. The variance of obtained rules from nursery dataset.

and assigned to first gene of chromosome, then, randomly, one gene is selected and its value is changed.

3.6. Final population selection operator

We explain this operator in Section 2.2. We use from Tournament Selection 2. In this strategy, first, a certain number of chromosomes are selected in basis of their fitness and then the best chromosome from them is spotted in next population (Eiben & Smith, 2003) the semi code of this function is showed in Fig. 7.

First, five chromosomes from parents' population and five chromosomes from offspring population are selected by *selectrandomchr* function and then the best ten selected chromosomes is selected by *get* & *removebestchr* function and is added to next population. Until size of population is less than the maximum population size, this action is repeated. The *selectrandomchr* function works as following: first, the population is sorted in descending order and is divided to four equal groups. The selection probability of first, second, third and fourth groups are, respectively, 0.1, 0.2, 0.3 and 0.4. After selection of one group, one chromosome is selected from it and this action (select one group and one chromosome from it) is repeated five times.

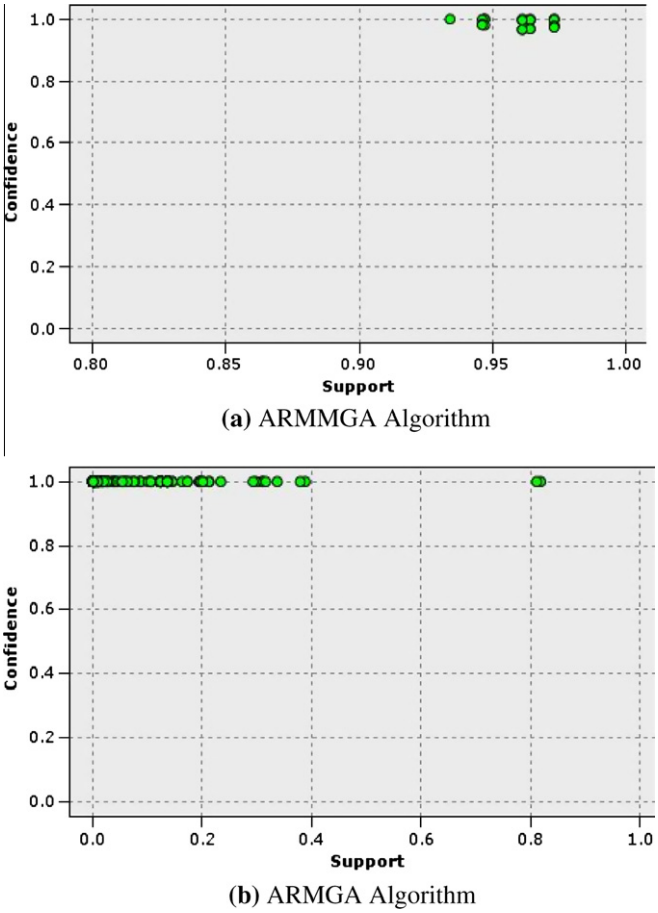
If one of parent or offspring population is finished, until the other population is finished or the size of next population is reached to the maximum population size, the action is repeated.

In the function, to prevent from staying on local minimum, the best three chromosomes of offspring population are selected and added to next population. The reason of this action is that in the crossover or mutation operator, some rare high-fitness chromosomes are produced such that the probability of their selection is small in Tournament Selection 2 strategy, so they are added to next population to prevent from staying our method on local minimum.

3.7. Initialization

We explain this operator in Section 2.2. ARMGA Initialization (Yan et al., 2008) produces other chromosomes base on seed. In it, first, seed chromosome is added to initial population and then the mutation function which its mutation probability is one, is applied on seed chromosome and the new produced chromosome is added to initial population. Until size of initial population is equal to or less than the half of maximum size of population, this operation is repeated on initial population. This approach has tow disadvantages. The first disadvantage is that the population is created based on seed chromosome and it may increase the probability that the variety of chromosomes decreases in initial population whereupon the probability of staying in local minimum are increased. The second disadvantage is that if the maximum size of population is not power of 2, the number of chromosome in initial population and thereupon, the probability of hasty converge and staying on local minimum are increased.

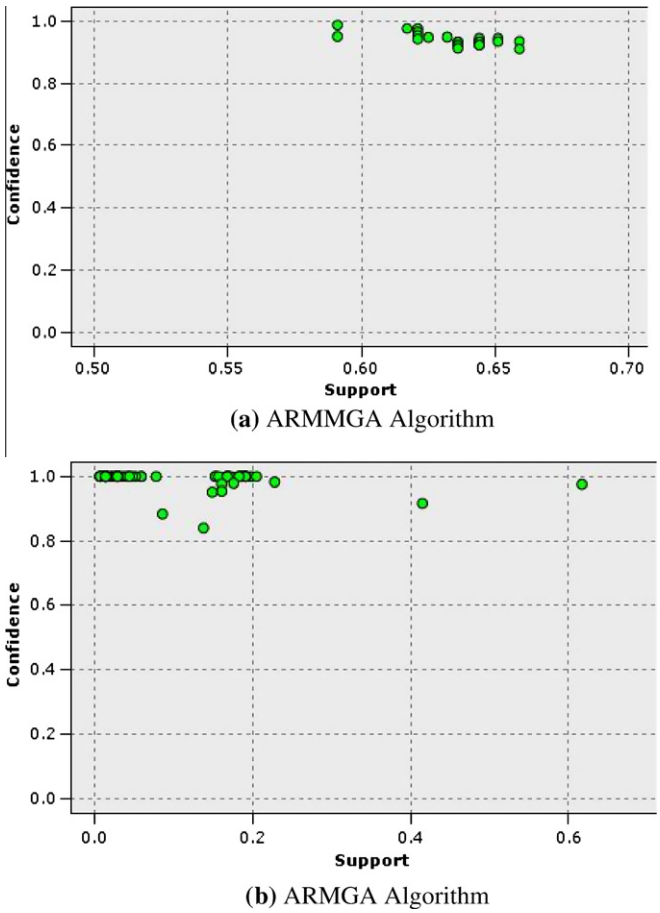
In our method, first we create a list of all items. Then we generate a random number between 1 to $k-1$ (the rule length) as a separator the antecedent from the consequent of rules and assign it at the first gene. If the size of items list is equal to or less than k, we randomly select k items from it and add to chromosome and then remove them from it, otherwise, first, we generate two list that one contains the items which exist in items list (*existList*) and other is



(a) ARMMGA Algorithm

(b) ARMGA Algorithm

Fig. 13. The variance of obtained rules from Solar Flare dataset.



(a) ARMMGA Algorithm

(b) ARMGA Algorithm

Fig. 14. The variance of obtained rules from SPECT heart dataset.

vice versa (*noExistList*). Then we add all items of *existList* to chromosome. The other items are selected randomly from *noExistList* and add them to chromosome and remove them from it. Finally, we combine the survivor items of *noExistList* with the items of *existList* and use it for generating of next chromosome. The generated chromosome is added to initial population and this operation is repeated until the size of initial population reaches to the maximum size of population. The semi code of it is showed in Fig. 8.

3.8. ARMMGA algorithm

This algorithm is the main cycle of genetic algorithm. Its parameters are selection probability, crossover probability and mutation probability and it returns the best produced population. The semi code of it is showed in Fig. 9.

In this semi code, first the initialization population is produced by initialization function and is spotted as the best population. Then if the termination function returns true, the best population is returned as output function, otherwise the selection function is performed on population and the parents' population is selected by it. Then the crossover function is performed on the parents' population and an offspring population is produced. In next step, the mutation function is performed on some offspring chromosomes and finally, the final population is selected from parent and offspring populations. If the average of fitness of final population is larger than the best population of previous stages or if they are equal and the variance of it is larger than or equal to the variance

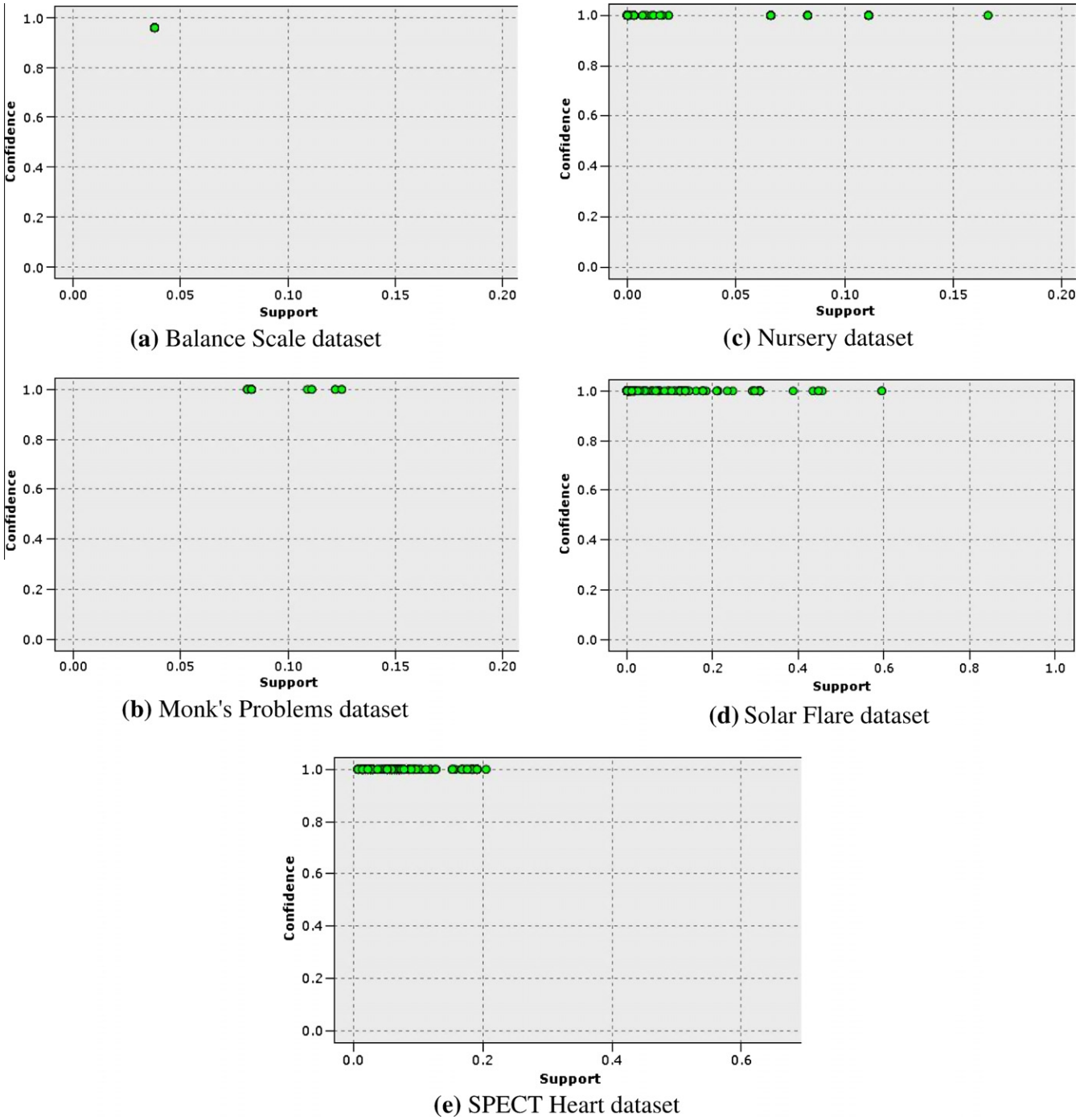


Fig. 15. The variance of produced rules on datasets by ARMMGA algorithm with fitness (8).

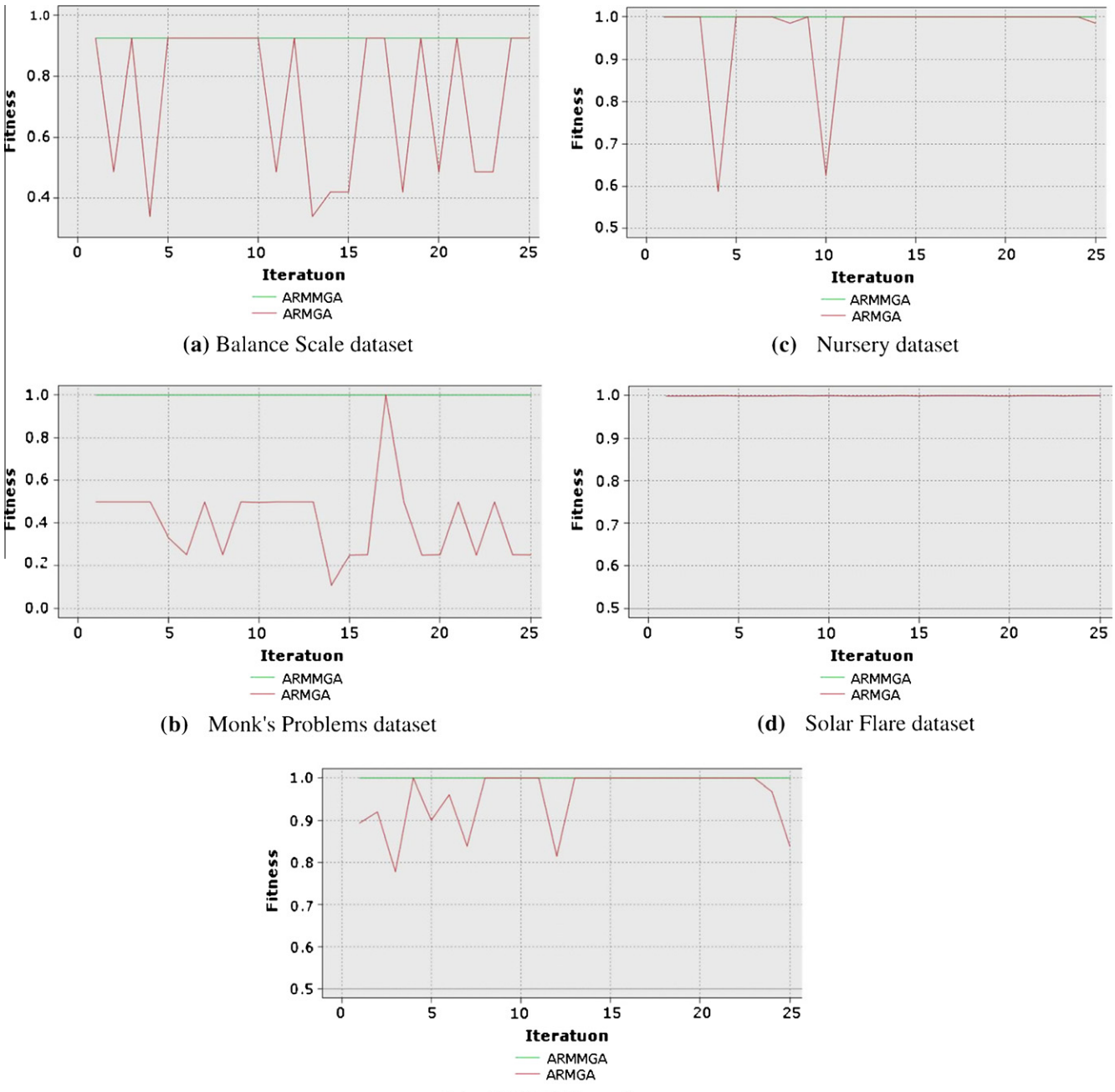
of fitness of the best population of previous stages, the best population is replaced with the current final population. The reason of this action is that because the goal is found the best population, not the best chromosome and after some cycles, the algorithm may reach to the best chromosome but the population has not a necessary convergence, so the action is repeated until the termination condition returns true value. Among these stages, rage chromosomes that have fitness smaller than the chromosomes of previous cycle, may be engendered in one cycle and whereas repetitive rules in population do not import for our, so the average and variance of not repetitive chromosomes is compared with the best population, if they better than the best population, it is spotted as the best population.

The ARMMGA algorithm terminates if one of the following cases happens.

- The difference between the best and the worst chromosomes is smaller than the value that is specified by user and the number of repetition of cycle is larger than the minimum value. The minimum of number of cycle is used because if the initialization population have chromosomes with low and close finesses, the algorithm is not terminated and continues its work.
- The number of repetition of cycle reached to the maximum value.

4. Tests and experiments

For testing the suggested algorithm and comparing with ARMGA algorithm (Yan et al., 2008), we use five datasets in UCI at http://www.ics.uci.edu/~mlearn. The specifications of them are implied in Table 1. The attributes of them are categorical, therefore, like.



(a) Balance Scale dataset

(c) Nursery dataset

(b) Monk's Problems dataset

(d) Solar Flare dataset

(e) SPECT Heart dataset

Fig. 16. The fitness average of final population at each 25 repetitions of run of ARMGA and ARMMGA in datasets.

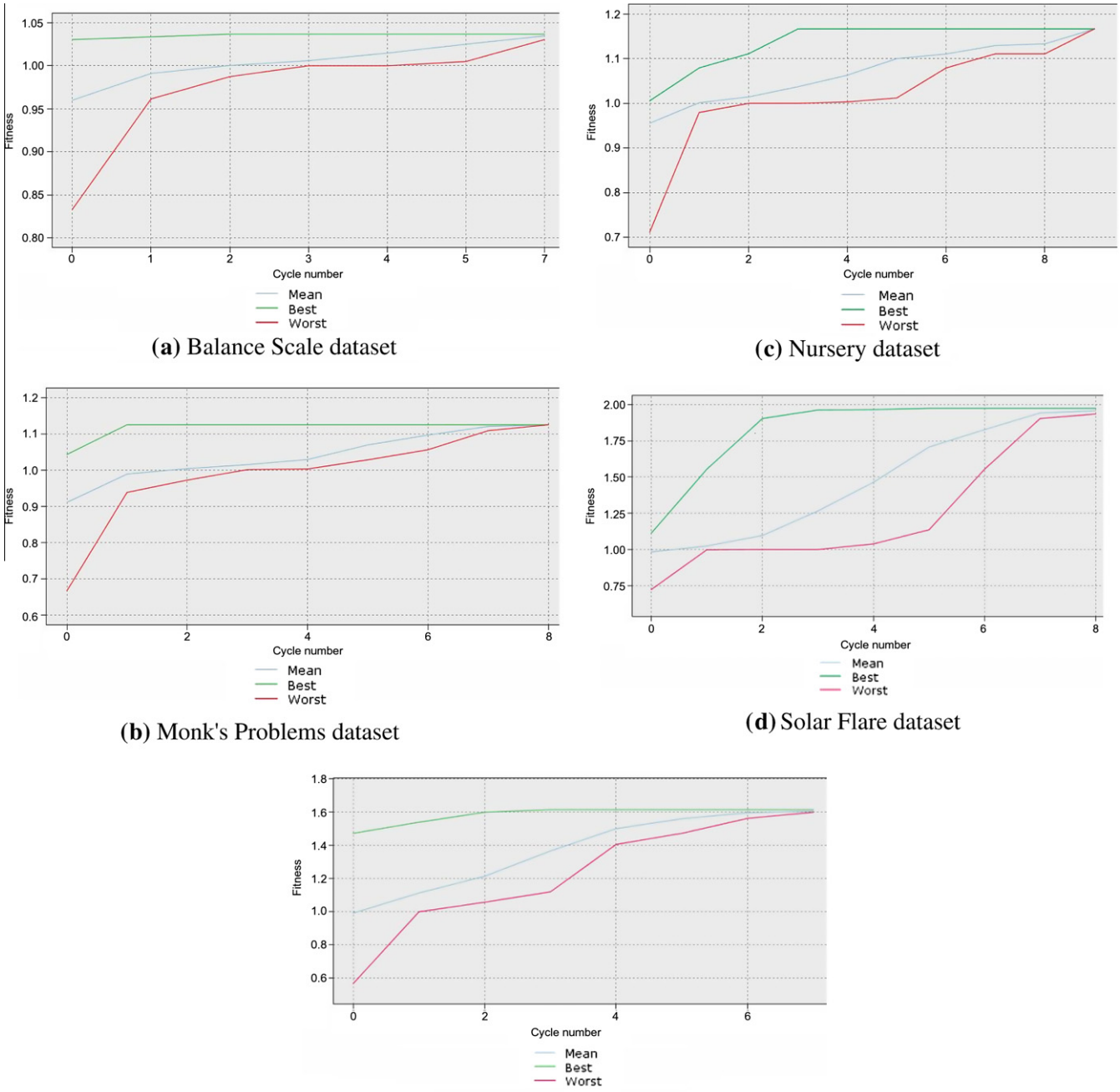
Paper (Yan et al., 2008), we convert them to Boolean datasets. We spot the attributes and each value as items such that one attribute with its value that exist in a record is assigned 1 and the attribute with each of other values are assigned 0 and one record includes items that their value is 1. Tables 2 and 3 are showed the parameters of two algorithms when we run them. In these tables, db is difference boundary between the best and worst chromosome, mi is the number of iteration of algorithms and si is the number of iteration of cycles in each algorithm.

Two algorithms are implemented with Java and are rune on computer with CPU 2000 Celeron Intel and 256 M Ram. To Speed to runs, we use the FP-tree to compute the fitness.

At paper (Yan et al., 2008), no function comes for selection of next population, so we use the suggested function of this article. Also, the Eq. (8) is used as fitness on it that its range are $(-\infty, 1]$

$$\frac{\sup(X\ U\ Y) - \sup(X)\sup(Y)}{\sup(X)(1-\sup(Y))}. \tag{8}$$

In suggested semi code, the expression *fitness* (chr) $*$ *frand* () < sp or cp or mp is used for selection, crossover and mutation, respectively. This expression further selects the chromosomes that have negative or low positive fitness that it is not logical. Therefore, we use the *normfitness* (chr) $*$ *frand* () $\geqslant$ 1-sp because the high positive values of fitness are selected with high probability. And we use the *normfitness* function because it changes the range of fitness to [0,1], so the negative fitness may be selected. This function only changes the values that are larger than -4 and ignores that values that are smaller than -4 because those almost do not import for our and the goal is reaching to chromosomes that their fitness is 1. The number of obtained rules is showed in Table 4.



(a) Balance Scale dataset

(c) Nursery dataset

(b) Monk's Problems dataset

(d) Solar Flare dataset

(e) SPECT Heart dataset

Fig. 17. The convergence of ARMMGA in datasets.

Also the Figs. 10–14 show the variance of rules according to support and confidence.

These figures show that ARMMGA Algorithm further finds the high support and confidence rules while the ARMGA Algorithm finds the high confidence and low support rules. Also number of Rules that obtained from ARMMGA in four datasets are very smaller than ARMGA Algorithm and we can decrease or increase them with decreasing or increasing the difference between the best and worst chromosome (db), or the number cycle of ARMMGA algorithm (si).

In ARMGA, because of its fitness function many rules might be obtain (Yan et al., 2008) but the suggested fitness function solves this problem and also because of the crossover strategy, the ARMGA might be stay on local minimum in some datasets. This problem clearly is showed in Figs. 10–12.

The suggested algorithm is not sensitive on the values of fitness function and these values only specify the precedence of chromosomes for running the algorithm genetic operators. This feature can be a positive characteristic and we obtain our ideal chromosomes with the related normal fitness, therefore in new test, we change the fitness function to Eq. (8) and obtain the following result.

The variance of obtained rules with change the fitness function is showed in Fig. 15. The rules that are obtained by local minimum are not seen in these pictures. We bring the fitness average diagrams of final population at each 25 repetition of ARMGA and ARMMGA Algorithms in Fig. 16 to show better the escape from local minimum by ARMMGA. The green lines are the average of fitness population of ARMMGA and red lines are the average of fitness population of ARMGA. The figures show that the ARMGA stays on local minimum in all datasets except Solar Flare.

The convergence of ARMMGA (without changing the fitness function) is showed in Fig. 17. The green lines are showed the fitness of best chromosome; the blue lines are showed the fitness average of population and the red lines are showed the fitness of worst chromosomes in each cycle. Because for each dataset 25 times of ARMMGA has been rune, therefore, we select the average of number cycles in each dataset. The figures show that the average of fitness in population becomes optimum in soft gradient.

5. Conclusions

In this paper, we present a total method with use of Genetic Algorithm for discovering association rules. The specifications of it are:

- This Genetic Algorithm obtain the optimum population not only the best chromosome
- To prevent from creation of invalid chromosomes in ARMGA, new crossover and mutation operators are presented.
- By suggestions such as the selection of best population in each repetition, the use of order of chromosomes in population as a criterion of their selection for running of Genetic Algorithm on them, the selection of several best chromosomes at offspring population for next population, the minimum number of cycle repetition and order-1 crossover, staying on local minimum is prevented. It is one of ARMGA problems.
- The minimum support and minimum confidence are not specified by user.
- One of ARMGA problems is the production of many rules (Yan et al., 2008). In ARMMGA, by presentation of new fitness function, this problem is solved.
- The most important of this algorithm is that the value of fitness function only specifies the order of chromosomes in population and has not any other effect on genetic algorithm operator (except termination function), therefore, this algorithm is convergence with any usually arbitrarily fitness function. The experiments show that the values of sp = 0.95, cp = 0.85 and mp = 0.01 are useful for the convergence of algorithm and with changing db (the difference between the best and worst chromosomes) and specifying the maximum number of cycles, we can reach to ideal results.

For improving the suggested algorithm, several works such as the obtaining of rules with variant length and the generalization of algorithm on categorical and numeric datasets can be stated.

Acknowledgements

We are thankful to our colleagues in Data Mining Laboratory in Computer Engineering Department at Iran University of Science and Technology for their cooperation and support. This work is also partially supported by Data and Text Mining Research Group at Computer Research Center of Islamic Sciences, NOOR Co. Tehran, Iran.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In *Proceedings of the 20th international conference on very large databases, Santiago, Chile*.

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD conference on management of data* (pp. 207–216).

Alatas, G., & Akin, E. (2007). Multi-objective differential evolution algorithm for mining numeric association rules. *Applied Soft Computing, 8*(1), 646–656.

Alatas, G., & Akin, E. (2008a). Rough particle swarm optimization and its applications. *Soft Computing, 12*, 1205–1218.

Alatas, G., & Akin, E. (2008b). Chaos rough particle swarm optimization and its applications. *Information Sciences, 163*, 194–203.

Chen, C.-H., Hong, T.-P., & Tseng, Vincent S. (2006). A cluster-based fuzzy-genetic mining approach for association rules and membership functions. In *IEEE*.

Chiu, H.-P., Tang, Y.-T., & Hsieh, K.-L. (2008). A cluster-based method for mining generalized fuzzy association rules. *IEEE Transactions on Fuzzy Sets and Systems, 16*(1), 249–262.

Davis, L. (Ed.). (1991). *Handbook of genetic algorithm*. New York: Van Nostrand Reinhold.

Dehuri, B., & Ghosh, A. (2004). Muti objective association rule mining using genetic algorithm. *Information Sciences, 163*, 123–133.

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing*. Springer.

Esmaeli, A., Nasiri, M., Minaei, B., & Mozayani., N. (2008). A method for association rule mining with PSO based on confodence. In *17th Conference of Electronic (ICEE)* (pp. 120–124).

Hadian, A., Nasiri, M., & Minaei-Bidgoli, B. (2009). Clustering based multi-objective rule mining using genetic algorithm. *International Journal of Digital Content Technology and its Applications, 4*(1), 37–42.

Houtsma, A., & Swami, M. (1993). *Set-oriented mining of association rules*. Research Report.

Kayaa, M., & Alhajj, R. (2005). Genetic algorithm based framework for mining fuzzy association rules. *Fuzzy Sets and Systems, 152*(3), 587–601.

Lin, D. I., & Kedem, Z. M. (1998). Pincer-search: An efficient algorithm for discovering the maximal frequent set. In *Proceedings of sixth European conference on extending database technology*.

Nasiri, M., Tagavi, L., & Minaei-Bidgoli, B. (2010). Numeric association rule mining using simulated anealing algorithm. *Journal of Convergence Information Technology, 5*(1), 60–68.

Tsay, Y. J., & Chiang, J. Y. (2005). CBAR: An efficient method for mining association rules. *Knowledge-Based Systems*, 99–105.

Yan, X., Zhang, C., & Zhang, Sh. (2008). Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support. *Expert Systems with Applications, 36*(2), 3066–3076.