An Efficient GA-Based Algorithm for Mining Negative Sequential Patterns

Zhigang Zheng[1], Yanchang Zhao[1,2], Ziye Zuo[1], and Longbing Cao[1]

[1] Data Sciences & Knowledge Discovery Research Lab
Centre for Quantum Computation and Intelligent Systems
Faculty of Engineering & IT, University of Technology, Sydney, Australia
{zgzheng,zzuo,lbcao}@it.uts.edu.au
[2] Centrelink, Australia
yanchang.zhao@centrelink.gov.au

Abstract. Negative sequential pattern mining has attracted increasing concerns in recent data mining research because it considers negative relationships between itemsets, which are ignored by positive sequential pattern mining. However, the search space for mining negative patterns is much bigger than that for positive ones. When the support threshold is low, in particular, there will be huge amounts of negative candidates. This paper proposes a Genetic Algorithm (GA) based algorithm to find negative sequential patterns with novel crossover and mutation operations, which are efficient at passing good genes on to next generations without generating candidates. An effective dynamic fitness function and a pruning method are also provided to improve performance. The results of extensive experiments show that the proposed method can find negative patterns efficiently and has remarkable performance compared with some other algorithms of negative pattern mining.

Keywords: Negative Sequential Pattern, Genetic Algorithm, Sequence Mining, Data Mining.

1 Introduction

The concept of discovering sequential patterns was firstly introduced in 1995 [1], and aimed at discovering frequent subsequences as patterns in a sequence database, given a user-specified minimum support threshold. Some popular algorithms in sequential pattern mining include AprioriAll [1], Generalized Sequential Patterns (GSP) [10] and PrefixSpan [8]. GSP and AprioriAll are both Apriori-like methods based on breadth-first search, while PrefixSpan is based on depth-first search. Some other methods, such as SPADE (Sequential PAttern Discovery using Equivalence classes)[12] and SPAM (Sequential PAttern Mining)[4], are also widely used in researches.

In contrast to traditional positive sequential patterns, negative sequential patterns focus on negative relationships between itemsets, in which, absent items are taken into consideration. We give a simple example to illustrate the difference: suppose p_1=<a b c d> is a positive sequential pattern; p_2=<a b $\neg c$ e> is a negative sequential pattern; and each item, a, b, c, d and e, stands for a claim item code in the customer claim database

M.J. Zaki et al. (Eds.): PAKDD 2010, Part I, LNAI 6118, pp. 262–273, 2010.

of an insurance company. By getting the pattern p_1, we can tell that an insurant usually claims for a, b, c and d in a row. However, only with the pattern p_2, we are able to find that given an insurant claim for item a and b, if he/she does NOT claim c, then he/she would claim item e instead of d. This kind of patterns cannot be described or discovered by positive sequential pattern mining.

However, in trying to utilize traditional frequent pattern mining algorithms for mining negative patterns, two problems stand in the way. (1) Huge amounts of negative candidates will be generated by classic breath-first search methods. For example, given 10 distinct positive frequent items, there are only 1,000 (=10^3) 3-item positive candidates, but there will be 8,000 (=20^3) 3-item negative candidates because we should count 10 negative items in it. (2) Take a 3-item data sequence $<a\ b\ c>$, it can only support candidates $<a>$, $<b>$, $<c>$, $<a\ b>$, $<a\ c>$, $<b\ c>$ and $<a\ b\ c>$. But in the negative case, data sequence $<a\ b\ c>$ not only supports the positive candidates as the above, but also can match a large bunch of negative candidates, such as $<a\ \neg a>$,$<b\ \neg a>$,$<b\ \neg b>$, $<a\ \neg a\ c>$, $<a\ \neg c\ c>$ etc. There are thus still huge amounts of negative candidates even after effective pruning.

Based on Genetic Algorithm (GA) [5], we propose a new method for mining negative patterns. GA is an evolvement method, which simulates biological evolution. A generation pass good genes on to a new generation by crossover and mutation, and the populations become better and better after many generations. We borrow the ideas of GA to focus on the space with good genes, because this always finds more frequent patterns first, resulting in good genes. It is therefore more effective than methods which treat all candidates equally, especially when a very low support threshold is set. It is equally possible to find long negative patterns at the beginning stage of process.

Our contributions are:

- A GA-based algorithm is proposed to find negative sequential patterns efficiently. It obtains new generations by crossover and mutation operations without generating candidates, and uses dynamic fitness to control population evolution. A pruning method is also provided to improve performance.
- Extensive experimental results on 3 synthetic datasets and a real-world dataset show that our algorithm has better performance compared with PNSP[11] and NegGSP[14] especially when the support threshold min_sup is very low.

This paper is organized as follows. Section 2 talks about related work. Section 3 briefly introduces negative sequential patterns and presents formal descriptions of them. Our GA-based algorithm is then described in Section 4. Section 5 shows experimental results on some datasets. The paper is concluded in the last section.

2 Related Work

Most research on sequential patterns has focused on positive relationships. In recent years, some research has started to focus on negative sequential pattern mining.

Zhao et al. [13] proposed a method to find negative sequential rules based on SPAM [4]. However the rules are limited to formats such as $<A \Rightarrow \neg B>$, $<\neg A \Rightarrow B>$, $<\neg A \Rightarrow \neg B>$. Ouyang & Huang [7] extended traditional sequential pattern definition

(A,B) to include negative elements such as *(¬A,B)*, *(A,¬B)* and *(¬A,¬B)*. They put forward an algorithm which finds both frequent and infrequent sequences and then obtains negative sequential patterns from infrequent sequences. Nancy et al. [6] designed an algorithm PNSPM and applied the Apriori principle to prune redundant candidates. They extracted meaningful negative sequences using the interestingness measure; nevertheless the above works defined some limited negative sequential patterns, which are not general enough. Sue-Chen et al. [11] presented more general definitions of negative sequential patterns and proposed an algorithm called PNSP, which extended GSP to deal with mining negative patterns, but they generated negative candidates in the appending step, which then may produce a lot of unnecessary candidates.

Some existing researches have used GA for mining the negative association rule and positive sequential pattern. Bilal and Erhan [2] proposed a method using GA to mine negative quantitative association rules. They generated uniform initial population, and used an adaptive mutation probability and an adjusted fitness function. [9] designed a GA to mine generalized sequential patterns, but it is based on SQL expressions. It is an instructive work since there are few research works using GA for negative sequential pattern mining.

3 Problem Statement

3.1 Definitions

A $sequence$ s is an ordered list of elements, $s = <e_1\ e_2\ ...\ e_n>$, where each e_i, $1 \leq i \leq n$, is an element. An $element$ e_i ($1 \leq i \leq$k) consists of one or more items. For example, sequence $<a\ b\ (c,d)\ f>$ consists of 4 elements and *(c,d)* is an element which includes two items. The $length$ of a sequence is the number of items in the sequence. A sequence with k items is called a *k-sequence* or *k-item sequence*.

Sequence is a general concept. We extend sequence definition to positive/negative sequence. A sequence $s=<e_1\ e_2\ ...\ e_n>$ is a $positive\ sequence$, when each element $e_i (1 \leq i \leq n)$ is a positive element. A sequence $s=<e_1\ e_2\ ...\ e_n>$ is a $negative\ sequence$, when $\exists i$, $e_i (1 \leq i \leq n)$ is a negative element, which represents the absence of an element. For example, $\neg c$ and $\neg(c,d)$ are negative elements, so $<a\ b\ \neg c\ f>$ and $<a\ b\ \neg(c,d)\ f>$ are both negative sequences.

A sequence $s_r=<e_{r_1}\ e_{r_2}\ ...\ e_{r_m}>$ is a $subsequence$ of another sequence $s_p=<e_{p_1}\ e_{p_2}\ ...\ e_{p_n}>$, if there exists $1 \leq i_1 \leq i_2 \leq ... \leq i_k \leq p_n$, $e_{r_1} \subseteq e_{p_{i_1}}$, $e_{r_2} \subseteq e_{p_{i_2}}$, ..., $e_{r_k} \subseteq e_{p_{i_k}}$.

A sequence s_r is a $maximum\ positive\ subsequence$ of another sequence s_p, if s_r is a subsequence of s_p, and s_r includes all positive elements of s_p. For example, $<a\ b\ f>$ is maximum positive subsequence of $<a\ b\ \neg c\ f>$ and $<a\ b\ \neg(c,d)\ f>$.

Definition 1: Negative Sequential Pattern. If the support value of a negative sequence is greater than the pre-defined support threshold *min_sup*, and it also meets the following constraints, then we call it a negative sequential pattern.

1) Items in a single element should be all positive or all negative. The reason is that a positive item and negative item in the same element are unmeaning. For example, $<a\ (a,\neg b)\ c>$ is not allowed since item a and item $\neg b$ are in the same element.

2) Two or more continuous negative elements are not accepted in a negative sequence. This constraint is also used by other researchers [11].

3) For each negative item in a negative pattern, its positive item is required to be frequent. For example, if $<\neg c>$ is a negative item, its positive item $<c>$ is required to be frequent. It is helpful for us to focus on the frequent items.

In order to calculate the support value of a negative sequence against the data sequences in a database, we need to clarify the sequence matching method and criteria. In other words, we should describe what kinds of sequence a data sequence can support.

Definition 2: Negative Matching. A negative sequence s_n=<e_1 e_2 ... e_k> $matches$ a data sequence s=<d_1 d_2 ... d_m>, iff:

1) s contains the max positive subsequence of s_n

2) for each negative element e_i*(1≤i≤k)*, there exist integers p, q, r*(1≤p≤q≤r≤m)* such that: $\exists e_{i-1} \subseteq d_p \wedge e_{i+1} \subseteq d_r$, and for $\forall d_q$, $e_i \not\subset d_q$

For example, see Table 1, s_n=<b $\neg c$ a> matches <b d a c>, but does not match <b d c a>, since the negative element c appears between the element b and a.

Table 1. Pattern matching

Pattern	match	Sequence
<b $\neg c$ a>	√	<b d a>
<b $\neg c$ a>	√	<b d a c>
<b $\neg c$ a>	×	<b d c a>

Table 2. Encoding

Sequence		Chromosome		
		$gene_1$	$gene_2$	$gene_3$
<a b $\neg$(c,d)>	⇒	+a	+b	$\neg$(c,d)

3.2 Ideas of GA-Based Method

As introduced in Section 1, negative sequential pattern mining may encounter huge amounts of negative candidates even after effective pruning. It will take a long time to pass over the dataset many times to get the candidates' support.

Based on GA, we obtain negative sequential patterns by crossover and mutation, without generating candidates; high frequent patterns are then selected to be parents to generate offspring. It will pass the best genes on to the next generations and will always focus on the space with good genes. By going through many generations, it will obtain a new and relatively high-quality population.

A key issue is how to find all the negative patterns since the GA-based method cannot ensure locating all of them. We therefore use an incremental population, and add all negative patterns, which are generated by crossover and mutation during the evolution process, into population. A dynamic fitness function is proposed to control population evolution. Ultimately, we can secure almost all the frequent patterns. The proportion can reach 90% to 100% in our experiments on two synthetic datasets.

4 GA-Based Negative Sequential Pattern Mining Algorithm

The general idea of the algorithm is shown as Fig. 1. We will describe the algorithm from how to encode a sequence, and then introduce population, selection, crossover, mutation, pruning, fitness function and so on. A detailed algorithm will then be introduced.

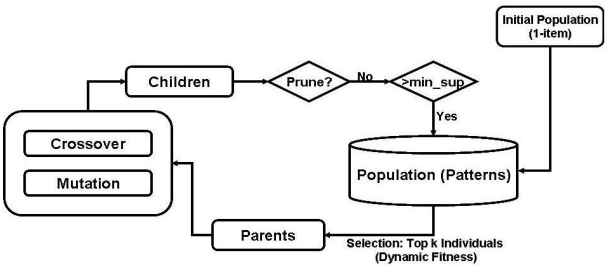

Fig. 1. Algorithm Flow

4.1 Encoding

Sequence is mapped into the chromosome code in GA. Both crossover and mutation operations depend on the chromosome code. We need to define the chromosome to represent the problems of negative sequential pattern mining exactly. There are many different methods to encoding the chromosome, such as binary encoding, permutation encoding, value encoding and tree encoding [5]. The permutation encoding method is suitable for ordering problem and its format is consistent with the format of the sequence data, so we use it for sequence encoding.

Each sequence is mapped into a chromosome. Each element of the sequence is mapped into a gene in the chromosome, no matter whether the element has one item or more. Given a sequence $<e_1\ e_2\ ...\ e_n>$, it is transformed to a chromosome which has n-genes. Each gene is composed of a tag and an element. The element includes one or more items, and the tag indicates that the element is positive or negative. For example, a negative sequence $<a\ b\ \neg(c,d)>$ is mapped into a 3-gene chromosome, see Table 2.

4.2 Population and Selection

In the classical GA method, the number of populations is fixed [5]. We using a fixed number of populations to produce the next generation, but the populations tended to contract into one high frequent pattern, and we can only obtain a small part of frequent patterns. To achieve as many sequential patterns as possible, we potentially needed a population to cover more individuals. We therefore adjusted the basic GA to suit negative sequential pattern mining in the following ways.

Initial Population. All 1-item frequent positive patterns are obtained first. Based on the 1-item positive patterns, we transform all of them to their corresponding 1-item negative sequences, such as transforming the frequent positive sequence $<e>$ to the negative sequence $<\neg e>$. We then take all positive and negative 1-item patterns as the initial population.

Population Increase. We do not limit population to a fixed number. When we acquire new sequential patterns during the evolvement, new patterns are put into the population for the next selection. If the population has already included the patterns, we ignore them. To improve the performance of this process, a hash table is used to verify whether a pattern has already appeared in the population.

Selection. The commonly used selection method is roulette wheel selection [3]. We have an increased population and the population number depends on the count of sequential patterns; thus, we can not use roulette wheel selection because the selection will be too costly if the population number is huge. We select the top K individuals with high dynamic fitness (see Section 4.5), where K is a constant number showing how many individuals will be selected for the next generation. To improve the performance of this selection method, we sort all individuals in population in descending order by dynamic fitness value. In every generation, we only select the first K individuals.

4.3 Crossover and Mutation

Crossover. Parents with different lengths are allowed to crossover with each other, and crossover may happen at different positions to get sequential patterns with varied lengths. For example, a crossover takes place at a different position, which is shown by '↕' in Table 3. After crossover, it may acquire two children. $Child1$ <b ¬c e> consists of the first part of $parent1$ and the second part of $parent2$. $Child2$ <d a> consists of the second part of $parent1$ and the first part of $parent2$. So we get two children with different lengths. If a crossover takes place both at the end/head of $parent1$ and at the head/end of $parent2$, as Table 4 shows, $child2$ will be empty. In that case, we shall set $child2$ by reverse. A $Crossover\ Rate$ is also used to control the probability of cross over when parents generate their children.

Table 3. Crossover

$parent1$	$b\ \neg c \updownarrow a$	⇒	$child1$	$b\ \neg c\ e$
$parent2$	$d \updownarrow e$	⇒	$child2$	$d\ a$

Table 4. Crossover at head/end

$parent1$	$b\ \neg c\ a \updownarrow$	⇒	$child1$	$b\ \neg c\ a\ d\ e$
$parent2$	$\updownarrow d\ e$	⇒	$child2$	$d\ e\ b\ \neg c\ a$

Mutation. Mutation is helpful in avoiding contraction of the population to a special frequent pattern. To introduce mutation into sequence generation, we select a random position and then replace all genes after that position with 1-item patterns. For example, given an individual <b ¬c a>, after mutation, it may change to <b d ¬e> if <d> and <¬e> are 1-item patterns. $Mutation\ Rate$ is a percentage to indicate the probability of mutation when parents generate their children.

4.4 Pruning

When a new generation is obtained after crossover and mutation, it is necessary to verify whether the new generation is valid in terms of the constraints for negative sequential patterns before passing over the whole dataset for their supports.

For a new individual c=<e_1 e_2 e_3 ... e_n>, c'=<e_i e_j ... e_k> ($0<i\leq j\leq k\leq n$) is the max positive subsequence of c, that is to say, e_i, e_j, ... and e_k are all positive elements, and other elements in c are negative. If c' is not frequent, c must be infrequent and should be pruned. This method is simple but effective for pruning invalid candidates without cutting off possible valid individuals by mistake.

4.5 Fitness Function

In order to evaluate the individuals and decide which are the best for the next generation, a fitness function for individuals is implemented in GA. We use the fitness function

shown in Eqn.(1):

$$ind.fitness = (ind.support - min\_sup) \times DatasetSize. \quad (1)$$

Fitness. The fitness function is composed of two parts. *Support* is the percentage that indicates how many proportion records are matched by the individual. If support is high, fitness will be high, so that the individual has good characteristics to pass down to next generation. *min_sup* is a threshold percentage value for verifying whether a sequence is frequent. *Dataset size* is the record count of whole dataset.

Dynamic Fitness. Because the characteristics of the individual have been transmitted to the next generation by crossover or mutation, the individual should exit after a few generations. The result will tend to contract to one point if the individual doesn't exit gradually. We therefore set a dynamic fitness $dfitness$ to every individual in the population, shown in Eqn.(2). Its initial value is equal to $fitness$, but decreases during the evolvement. It indicates that the individuals in the population will gradually ceased to evolve. It is like a life value. When an individual's dynamic fitness is low or close to 0($<$0.01), we set it to 0 because we regard it as a wasted individual which cannot be selected for the next generation.

$$ind.dfitness = \begin{cases} ind.fitness, & \text{initial set} \\ ind.dfitness \times (1 - DecayRate), & \text{if } ind \text{ is selected} \end{cases} \quad (2)$$

Decay Rate. We set a decay rate to indicate the decrease speed of individual's fitness. The decay rate is a percentage value between 0% and 100%. If an individual is selected by the selection process, its dynamic fitness will decrease by the speed of the decay rate. If the decay rate is high, dynamic fitness will decrease quickly and individuals will quickly cease to evolve. Thus, we may get less frequent patterns through a high decay rate. If we want to obtain the maximum frequent patterns, we can set a low decay rate, such as 5%, but this will give rise to a longer running time.

4.6 Algorithm Description

Our algorithm is composed of the following six steps.

Step 1:We obtain the initial population which includes all frequent 1-item positive and 1-item negative sequences. ***Step 2:*** Calculate all initial individuals' fitness. Their $dynamic\ fitness$ is set to their $fitness$. ***Step 3:*** We select the top K individuals with high dynamic fitness from the population. After selection, the dynamic fitness of the selected individuals is updated by Eqn.(2). ***Step 4:*** Crossover and mutation between the selected individuals to produce the next generation. ***Step 5:*** After obtaining the next generation, we first prune invalid individuals and then calculate the frequency and fitness of remained individuals in new generation. If the frequency of an individual is greater than *min_sup*, we add it into the population, and set its fitness and dynamic fitness, but if the population has included this individual, we ignore it. ***Step 6:*** Go back to step 3 and iterate the above process until all individuals in the population are dead (i.e., their dynamic fitness has become close to 0). The dead individuals are still in

population, but they ceased to evolve. In the end, we obtain the final result - whole population, which is composed of all dead individuals.

The pseudocode of our algorithm is given as follows.

```
RunGA(min_sup, decay_rate, crossover_rate, mutation_rate){
  pop = initialPopulation();
  for (each individual ind in pop){
    ind.fitness = calculateFitness(ind);
    ind.dfitness = ind.fitness
    pop.sum_dfitness = pop.sum_dfitness + ind.dfitness
  }
  while ( pop.sum_dfitness > 0 ){
    popK = Selection(pop);
    if (Random()<crossover_rate) Crossover(popK);
    if (Random()<mutation_rate) Mutation(popK);
    for (each individual ind in popK)
      if (Prune(ind) ! =true && ind.sup >= min_sup)    pop.add(ind);
  }
  return pop;
}

Selection(pop){ //Subfunction for selecting top K individuals from population
  for (each ind with top K dfitness in pop){
    popK.add(ind);
    ind.dfitness = ind.dfitness * (1-decay_rate);
    if (ind.dfitness < 0.01) ind.dfitness = 0;
  }
  return popK;
}
```

5 Experiments

Our algorithm was implemented with Java and tested with three synthetic sequence datasets generated by an IBM data generator [1] and a real-world dataset. We also implemented the PNSP algorithm [11] and Neg-GSP algorithm [14] with Java for performance comparison. All the experiments were conducted on a PC with Intel Core 2 CPU of 2.9GHz, 2GB memory and Windows XP Professional SP2.

$Dataset1(DS1)$ is C8.T8.S4.I8.DB10k.N1k, which means the average number of elements in a sequence is 8, the average number of items in an element is 8, the average length of a maximal pattern consists of 4 elements and each element is composed of 8 items average. The data set contains 10k sequences, the number of items is 1000.

$Dataset2(DS2)$ is C10.T2.5.S4.I2.5.DB100k.N10k.

$Dataset3(DS3)$ is C20.T4.S6.I8.DB10k.N2k.

$Dataset4(DS4)$ is real application data for insurance claims. The data set contains 479 sequences. The average number of elements in a sequence is 30. The minimum number of elements in a sequence is 1, and the maximum number is 171.

Experiments were done to compare the different $Crossover\ Rate$, $Mutation\ Rate$ and $Decay\ Rate$ on two synthetic datasets, $DS1$ and $DS2$. Each experiment was run 10 times and then the average value was got as the final result. We focused on comparing runtime, the number of patterns and the runtime per pattern, which indicates how long it takes to get one pattern. The total number of all patterns was determined by PNSP and

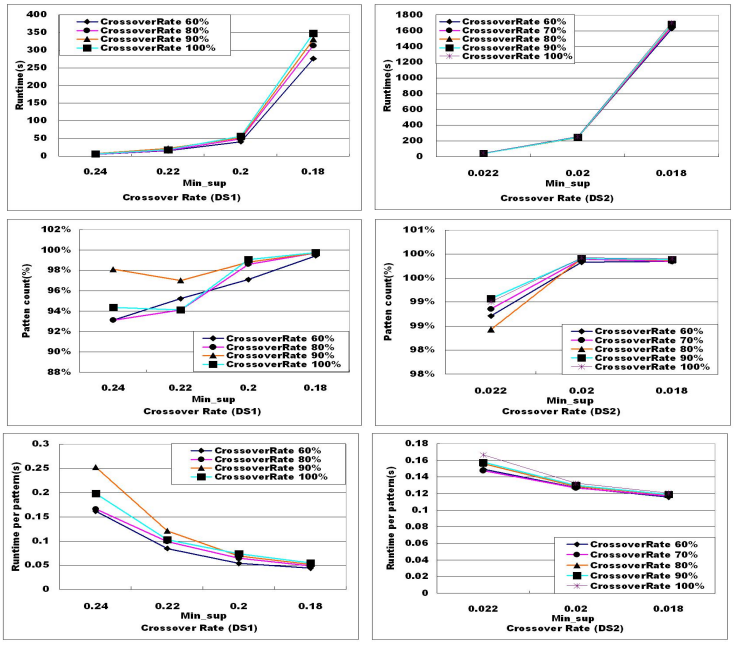

Fig. 2. Different Crossover Rates

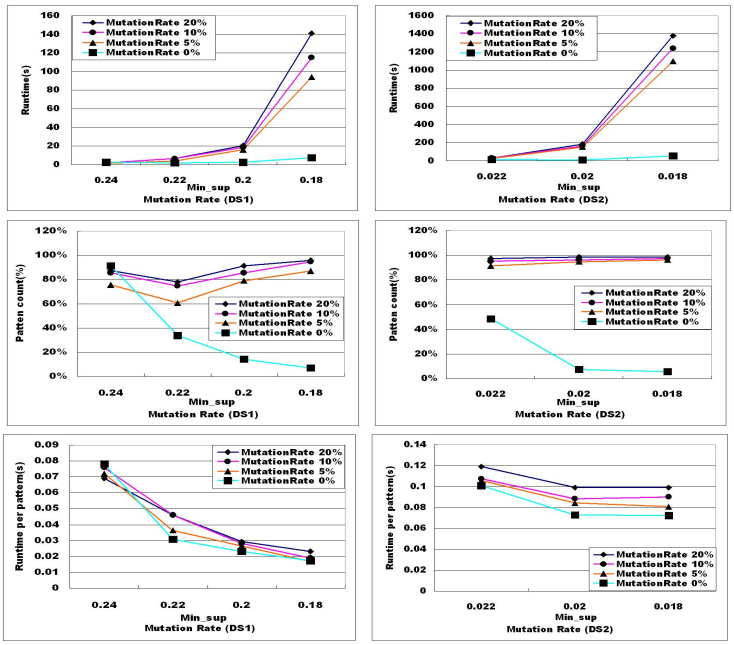

Fig. 3. Different Mutation Rates

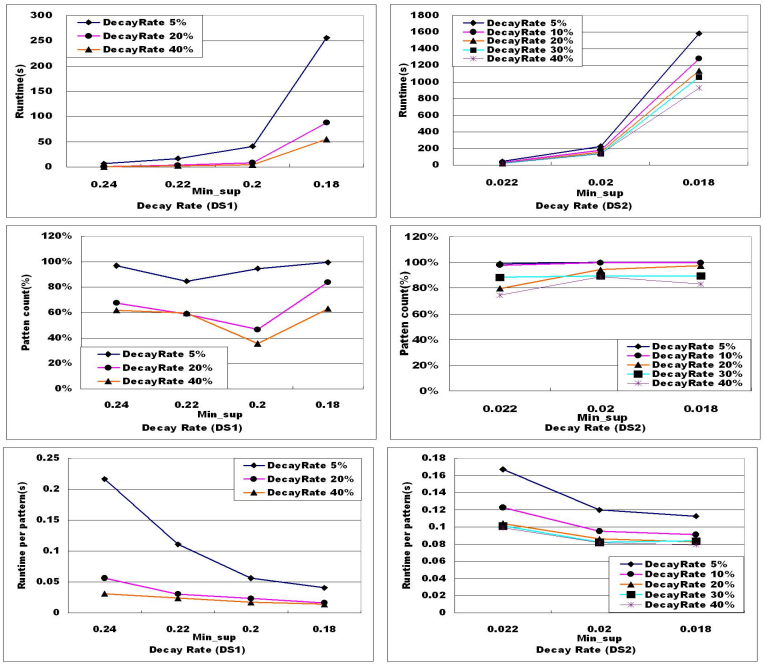

Fig. 4. Different Decay Rates

Neg-GSP algorithm, and it was then easy to know the proportion of patterns we could get by using our algorithm. The Y axis (see the 3rd and 4th chart of Fig. 2) indicates the proportion of patterns.

Crossover Rate. We compared different crossover rates from 60% to 100%. Fig. 2 shows the effect of different crossover rates on $DS1$ and $DS2$. With low crossover rates, such as 60%, we obtained almost the same proportion of patterns as with high crossover rates (see the 2nd and 5th charts in Fig. 2). The least runtime per pattern is achieved when the crossover rate is low, so 60% is the best choice for the two datasets in our experiments.

Mutation Rate. We compared different mutation rates from 0% to 20% on $DS1$ and $DS2$ (see Fig. 3). They show that the mutation rate will not have an outstanding effect, but if it is set to 0%, it will result in missing a lot of patterns. A Mutation rate of 5-10% is a good choice because it can produce around 80% patterns for $DS1$ and above 90% patterns for $DS2$. When the mutation rate is 5%, the average runtime per pattern is lower. We therefore set a mutation rate of 5% for the following experiments.

Decay Rate. Decay rate is a variable that we used to control evolution speed. If the decay rate is high, individuals will die quickly, so we can get only small proportion of patterns. If the decay rate is low, we can get more patterns, but a longer runtime is necessary (see Fig. 4). In order to get all negative sequential patterns, we always choose $decayrate$=5%, which enables us to obtain around 90% to 100% patterns on the two datasets.

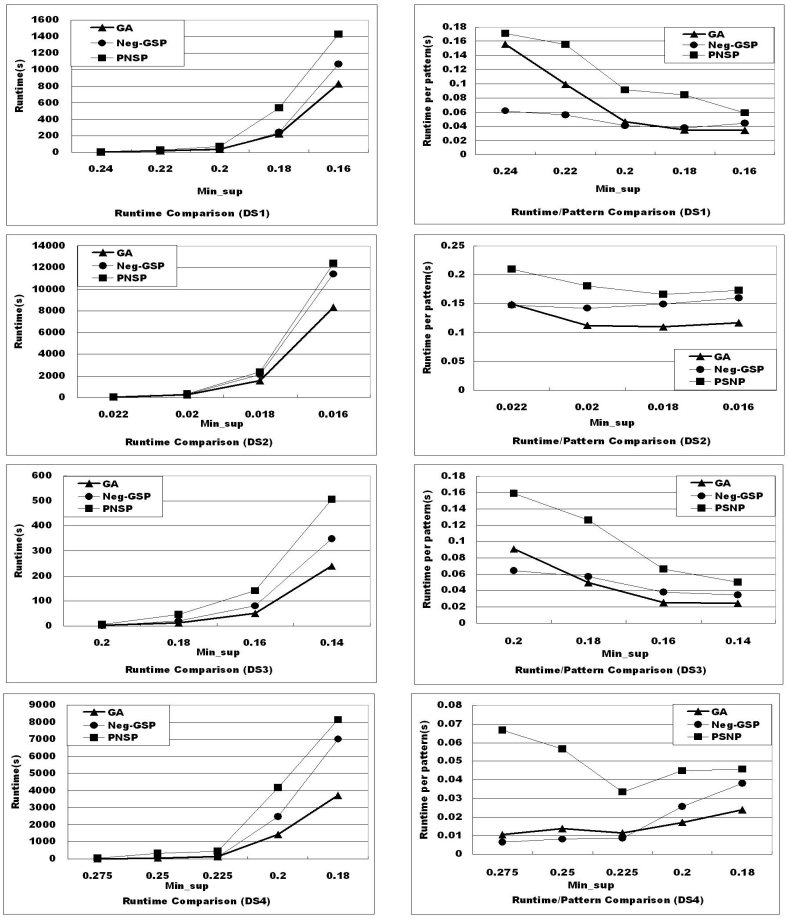

Fig. 5. Comparison with PNSP and Neg-GSP Algorithms

Performance Comparison. We compared our algorithm with PNSP and Neg-GSP, which are two algorithms proposed recently for negative sequential pattern mining. The tests are based on $Crossover\ Rate$=60%, $Mutation\ Rate$=5% and $Decay\ Rate$=5%. The results (see Fig. 5) on 4 different datasets show that the performance of the GA-based algorithm is better than PNSP and Neg-GSP when the support threshold is low. Our algorithm is not better than others when min_sup is high, because most patterns are very short and the GA-based method cannot demonstrate its advantage.

When min_sup is high, there are not as many patterns and the patterns are short, so it is very easy to find the patterns with existing methods. However, when min_sup is low, the patterns are longer and the search space is much bigger, so it is time-consuming to find patterns using traditional methods. Using our GA-based algorithm, it is still can obtain the patterns quickly even though min_sup is very low.

6 Conclusions and Future Work

Based on GA, we have proposed a method for negative sequential pattern mining. Extensive experimental results on synthetic datasets and a real-world dataset show that the proposed method can find negative patterns efficiently, and it is better than existing algorithms when the support threshold min_sup is low or when the patterns are long.

In our future work, we will focus on studying new measures including fitness function, selection and crossover method to make our algorithm more efficient. There should also be some better methods for pruning. Other work will be to explore post mining to find interesting patterns from the discovered negative sequential patterns. As we have obtained many negative sequential patterns, the means of finding interesting and interpretable patterns from them is valuable in industry applications.

References

1. Agrawal, R., Srikant, R.: Mining Sequential Patterns. In: Yu, P.S., Chen, A.L.P. (eds.) 11th International Conference on Data Engineering, pp. 3–14. IEEE Computer Soc. Press, Taipei (1995)
2. Bilal, A., Erhan, A.: An Efficient Genetic Algorithm for Automated Mining of Both Positive and Negative Quantitative Association Rules. Soft. Computing 10(3), 230–237 (2006)
3. Haupt, R.L., Haupt, S.E.: Practical Genetic Algorithms. Wiley, New York (1998)
4. Jay, A., Jason, F., Johannes, G., Tomi, Y.: Sequential PAttern Mining Using a Bitmap Representation. In: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, Edmonton (2002)
5. Mitchell, M.: Introduction to Genetic Algorithms. MIT Press, Cambridge (1996)
6. Nancy, P.L., Hung-Jen, C., Wei-Hua, H.: Mining Negative Sequential Patterns. In: Proceedings of the 6th Conference on WSEAS International Conference on Applied Computer Science, vol. 6, WSEAS, Hangzhou (2007)
7. Ouyang, W.M., Huang, Q.H.: Mining Negative Sequential Patterns in Transaction Databases. In: 2007 International Conference on Machine Learning and Cybernetics, vol. 2, pp. 830–834 (2007)
8. Pei, J., Han, J., Mortazavi-Asl, B., Jianyong, W., Pinto, H., Qiming, C., Dayal, U., Mei-Chun, H.: Mining Sequential Patterns by Pattern-growth: The PrefixSpan Approach. IEEE Transactions on Knowledge and Data Engineering 16, 1424–1440 (2004)
9. Sandra de Amo, A.d.S.R.J.: Mining Generalized Sequential Patterns Using Genetic Programming. In: ICAI 2003, Las Vegas, Nevada, USA (2003)
10. Srikant, R., Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In: Proceedings of the Fifth International Conference on Extending Database Technology, EDBT (1998)
11. Sue-Chen, H., Ming-Yen, L., Chien-Liang, C.: Mining Negative Sequential Patterns for E-commerce Recommendations. In: Proceedings of the 2008 IEEE Asia-Pacific Services Computing Conference, IEEE Computer Society Press, Los Alamitos (2008)
12. Zaki, M.J.: SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning 42, 31–60 (2001)
13. Zhao, Y., Zhang, H., Cao, L., Zhang, C., Bohlscheid, H.: Efficient Mining of Event-Oriented Negative Sequential Rules. In: IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, WI-IAT 2008, vol. 1, pp. 336–342 (2008)
14. Zheng, Z., Zhao, Y., Zuo, Z., Cao, L.: Negative-GSP: An Efficient Method for Mining Negative Sequential Patterns. In: The 8th Australasian Data Mining Conference. AusDM 2009. Data Mining and Analytics, Melbourne, Australia, vol. 101, pp. 63–67 (2009)