Proceedings of the 2007 IEEE Symposium on
Computational Intelligence and Data Mining (CIDM 2007)

GAIS: A Method for Detecting Interleaved Sequential Patterns from Imperfect Data

Marja Ruotsalainen
Institute of Signal Processing
Tampere University of Technology
Tampere, Finland.
Email: marja.ruotsalainen@tut.fi

Timo Ala-Kleemola
Institute of Signal Processing
Tampere University of Technology
Tampere, Finland.
Email: timo.ala-kleemola@tut.fi

Ari Visa
Institute of Signal Processing
Tampere University of Technology
Tampere, Finland.
Email: ari.visa@tut.fi

Abstract— **This paper introduces a novel method, GAIS, for detecting interleaved sequential patterns from databases. A case, where data is of low quality and has errors is considered. Pattern detection from erroneous data, which contains multiple interleaved patterns is an important problem in a field of sensor network applications. We approach the problem by grouping data rows with the help of a model database and comparing groups with the models. In evaluation GAIS clearly outperforms the greedy algorithm. Using GAIS desired sequential patterns can be detected from low quality data.**

I. Introduction

The widespread deployment of sensors in measurement, detection, and monitoring applications have created a need for processing of sequential data [1], [2]. In these sensor network applications, data usually streams continuously from various known sources (e.g. sensors) being imprecise, imperfect [3] and even lost. Because data streams have redundant information, it is not appropriate to store all the data but remove redundancy in preprocessing step. After removal sensor data can be presented as time series or some other type of sequential data [4].

Let us consider an example of a monitoring system which supervises a large machine at a factory. The machine has several integrated sensors, which measure the temperature, pressure, motion etc. in various parts of the machine. Measurements are imprecise, they can be out of sensor range and a sensor itself can break. Data stream from sensors is preprocessed and redundancy is removed in a sense that we are only interested in at least moderate changes in measurement values. Then data is converted into sequential form in which measurement values are categorized. The machine is supervised to predict fault situations. Before faults measurement values of physical quantities change in a certain way so we are able to create a database of sequential model patterns. Changes preceding various faults can overlap so data contains interleaved patterns.

If sequential data is represented as a sequence of characters or comparable attribute values, classical string matching approaches such as Boyer-Moore [5] or Knuth-Morris-Pratt [6] may be useful. These approaches detect patterns by comparing characters or attribute values in a pattern to those in data. For the more complex data, such as multidimensional, there are pattern matching techniques [7], which use more sophisticated comparison operations than basic string matching. In case of real world data, approximate string or pattern matching is often used [8]. Approximate sequential pattern matching is considered, especially, in time series similarity research [9], [10].

Recently, sequential patterns have been detected from data streams of sensors [11]. These applications usually have real time requirements and previous values are checked within a time window. Another interesting application area is user activity data. There sequential patterns have been used, for example, in anomaly detection [12]. However, research on low quality sequential data has been minor even though many real world applications produce it.

If the detection problem is converted into a combinatorial optimization problem, many heuristic search methods, such as simulated annealing [13], swarm intelligence (ant colony optimization [14], [15] and particle swarm optimization [16]), tabu search [17] and genetic algorithms [18], [19], can be used to solve it.

In this paper, we present a method called GAIS (a Genetic Algorithm based method for Interleaved Sequential pattern detection). Assume we have a database where each row contains information on one observation including registration time and measurements. Observations come from various sources and they can be long-lasting. Instead of starting and ending time, usually only registration time is available. Measurements can have missing values and existing values can be incorrect. The patterns we wish to detect, can contain data from any source and be temporally interleaved with each other.

Section II introduces event sequence formulation and related terms. Section III presents the principles of GAIS. Section IV documents the comparison of GAIS to the greedy algorithm. Discussion is in Section V and Section VI is the conclusion.

II. Event sequence formulation

Assume a database containing observations. After categorization, each observation can be represented as an event $e_k = (a_k, t_k)$, where a_k is a type and t_k is a time stamp. Type a_k is the categorization result and time stamp t_k the registration time of the observation, which corresponds to the event e_k.

```
Event sequence E
----------------
Event types: E X O A M N P L E E
Time stamps: 1 2 3 4 5 6 7 8 9 10

Subsequence B1
--------------
Event types: O N E
Time stamps: 3 6 10

Subsequence B2
--------------
Event types: E X A M P L E
Time stamps: 1 2 4 5 7 8 9

Subsequences B1 and B2 form a partition
P of the event sequence E.
```

Fig. 1. An example of an event sequence E and its two subsequences, B_1 and B_2. Subsequences form a partition of E, because union of B_1 and B_2 is E and intersection of B_1 and B_2 is empty.

An event sequence of length l is $E = e_1, e_2, \ldots, e_l, \quad t_k < t_{k+1}$. Empty sequence has length 0. A subsequence of an event sequence E is $B = e_{p_1}, e_{p_2}, \ldots, e_{p_m}, \quad t_{p_i} < t_{p_{i+1}}$ and a subsequence set is C. A subsequence set C of n subsequences forms a partition P of the event sequence E if $E = \bigcup_{i=1}^{n} B_i, \ \forall B \in C$ and $B_i \bigcap B_j = \emptyset, \ i \neq j$. A model $m = (E, u)$ is an event sequence E, which has an interpretation u. Models form a model set M.

Figure 1 shows an example of an event sequence and its two subsequences. The subsequences form a partition of the event sequence. For clarity, this and following examples use time stamps $1 \ldots n$.

We developed event histograms because we needed a way to express the event structure of an event sequence. In addition, event histograms were found useful in GA operations such as crossing and fitness value calculation. The event histogram of an event sequence describes how many times each type recurs. For example, the event histogram of the event sequence $(A, 1)(E, 2)(E, 3)(B, 4)(C, 5)(B, 6)$ is 12102 when types are in order $ABCDE$.

III. GAIS

This section introduces GAIS, a genetic algorithm based method for interleaved sequential pattern detection. Figure 2 shows the principles of GAIS in general. At first GAIS generates an initial population, which consists of randomly created individuals. Then the fitness value is calculated for each individual using models in the model set. After fitness calculation, individuals are selected, crossed and mutated. Crossing produces new individuals and these individuals form a new population. Elitism is in use. It is implemented in such a way that a randomly chosen new individual is replaced with the best individual of the previous population. Then all the operations of the iteration cycle are performed again beginning from fitness calculation. Iteration continues until the stopping criterion is met. Finally, decision is made for each chromosome using the model set.

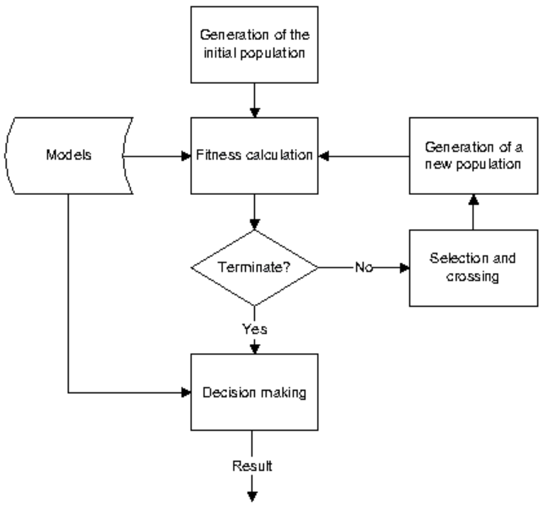


Fig. 2. The principles of GAIS in general. At first the initial population is generated, and then fitness calculation, selection, and crossing are performed. Iteration continues until the stopping criterion is met.

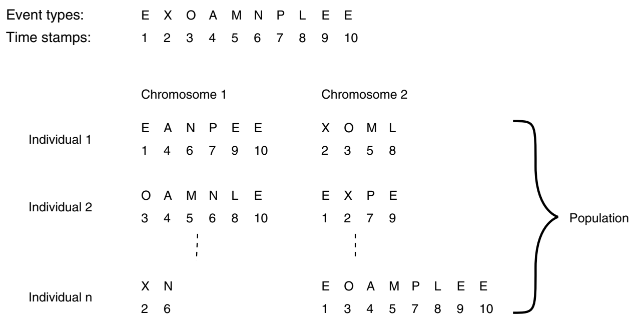


Fig. 3. An example of an initial population for an event sequence. Population contains n individuals, which consist in this example of two chromosomes.

A. Generation of the initial population

Initial population contains initial individuals. Individuals are solution candidates, which consist of chromosomes. Chromosomes are subsequences B_i of the event sequence E and chromosomes of an individual form a partition. An initial individual is formed by distributing all the events in E to the chromosomes randomly. So an individual contains all events of E once, and only them. Figure 3 shows an example of an initial population and clarifies terms. Population size and chromosome amount can be adjusted. Typical population size in genetic algorithm applications is from a few dozens to thousands [19].

B. Fitness calculation

Fitness values are calculated by comparing chromosomes, one at a time, with all models. Fitness of a chromosome is the distance from it to the closest model. Fitness of an individual is a weighted sum of its chromosomes' fitness values. Two distance measures are used to compare chromosomes. The first of those measures distance between event sequence representations and the second between event histograms. The fitness function is a sum of these measures.

Event sequence representations are compared using Edit distance [20]. It considers the difference in time order of events. Edit distance is calculated from types. Comparison of event histograms is done using Manhattan distance (1. norm). If corresponding values in the histogram differ from each other, the distance is added by the absolute value of the difference.

Equation 1 shows the fitness function f. A partition is P_k and a model set is M. Function $Edit(B_i, m_j)$, where a subsequence $B_i \in P_k$ and a model $m_j \in M$, computes Edit distance from a subsequence B_i to a model m_j. Function $L_1(B_i, m_j)$ computes the Manhattan distance from a subsequence B_i to a model m_j. The number of subsequences B is n.

$$f(P_k, M) = \sum_{i=1}^{n} \alpha_i \min_j (Edit(B_i, m_j) + L_1(B_i, m_j)) \quad (1)$$

For the largest distance the weight coefficient α is 1, for the second largest it is 2 and so on. For the smallest distance, α is n. Variable α is used, because a partition consisting of good and poor subsequences is more desirable than a partition consisting of average ones.

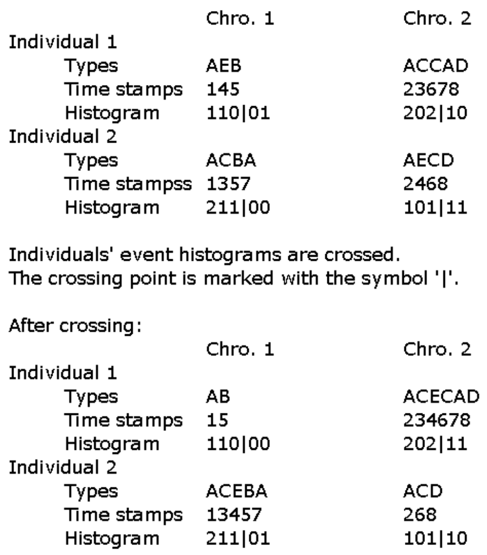
C. Selection and crossing

Selection part of GAIS implements Stochastic Universal Sampling with linear ranking selection [21]. After selection, GAIS forms pairs of selected individuals randomly and crosses them. Crossing is carried out for the event histograms of the individuals. At first, each chromosome is cut at the same randomly selected point. Then the parts after the crossing point are switched between individuals. The crossing point is the same for each chromosome of an individual pair, but differs from pair to pair.

Figure 4 illustrates the crossing of two individuals. Assume an event sequence $(A,1)(A,2)(C,3)(E,4)(B,5)(C,6)(A,7)(D,8)$ and two selected individuals consisting of two chromosomes each. Individuals are shown in Figure 4. At first, event histograms (event order $ABCDE$) are calculated, and the crossing point (the value of a random number generator) is chosen. Then crossing is performed resulting in two new individuals.

D. Mutation

After crossing, GAIS mutates new individuals. A mutation rate, a probability for an individual to mutate, can be adjusted. Mutation is either swap mutation or relocation mutation. We have developed swap mutation to shape the event histogram



```
                      Chro. 1           Chro. 2
Individual 1
        Types         AEB               ACCAD
        Time stamps   145               23678
        Histogram     110|01            202|10
Individual 2
        Types         ACBA              AECD
        Time stampss  1357              2468
        Histogram     211|00            101|11

Individuals' event histograms are crossed.
The crossing point is marked with the symbol '|'.

After crossing:
                      Chro. 1           Chro. 2
Individual 1
        Types         AB                ACECAD
        Time stamps   15                234678
        Histogram     110|00            202|11
Individual 2
        Types         ACEBA             ACD
        Time stamps   13457             268
        Histogram     211|01            101|10
```

Fig. 4. Illustration of the crossing operation. Crossing is performed for event histograms. The crossing point is marked with the symbol '|'.

of an individual and relocation mutation to shape the event sequence representation.

1) Swap mutation: In swap mutation one event from one chromosome is chosen randomly. Then it is swapped with another randomly chosen event that has the same type and is located in some other chromosome (if an event meeting these criteria exists). After swap mutation, each chromosome has the same types as they had before it, but the order of events may be different. This mutation type is useful, when the types in chromosomes are desired, but their order is wrong.

Assume, for example, an event sequence $(A,1)(A,2)(C,3)(E,4)(B,5)(C,6)(A,7)(D,8)$ and an individual with two chromosomes $(A,1)(E,4)(B,5)$ and $(A,2)(C,3)(C,6)(A,7)(D,8)$. One event is chosen randomly. Let it be, for example, $(A,7)$ in the second chromosome. Then $(A,7)$ is swapped with some event, which has the same type and is located in the other chromosome. That event is $(A,1)$. The result is then $(E,4)(B,5)(A,7)$ and $(A,1)(A,2)(C,3)(C,6)(D,8)$. Swap mutation does not change the histogram of an individual.

2) Relocation mutation: In relocation mutation, an event is chosen randomly and it is relocated into another chromosome. Relocation mutation modifies the histograms of chromosomes. It has an important role in the first few populations, where the event structure of the chromosomes is not settled yet.

Assume again chromosomes $(A,1)(E,4)(B,5)$ and $(A,2)(C,3)(C,6)(A,7)(D,8)$. When the event $(A,7)$ in the second chromosome is chosen and relocated into the first chromosome, the result is $(A,1)(E,4)(B,5)(A,7)$ and $(A,2)(C,3)(C,6)(D,8)$.

E. Empty model and decision making

The true number of patterns, which are interleaved in an event sequence, is not known. Therefore, chromosome amount cannot be set exactly, only rough upper estimates can be given. A specialized model, an empty model, is developed to overcome this situation. Using the fitness function in (1) the distance from a subsequence to the empty model is commonly smaller than the distance from a subsequence to any real model if they do not have the same event structure.

The empty model can be added into a model set at any given time. However, if it is added too early, results will suffer, because the empty model is the fittest for random subsequences of an event sequence. Therefore, the genetic algorithm must be allowed to form models like event structures for the subsequences before the empty model is added.

At the end of the execution, GAIS makes a decision for each chromosome. The decision is the model that fits the best with the chromosome. The empty model as a decision may indicate that a previously unseen model has been found.

IV. EXPERIMENTS

This section introduces the comparison of GAIS to the greedy algorithm. The greedy algorithm has been used in the following way:

1) The longest common subsequence is taken between the event sequence and each model.
2) Ratio "subsequence length per model length" is calculated for each model.
3) The model, which has the largest ratio, is considered. If the ratio is greater than or equal to 0.4, the corresponding subsequence is subtracted from the event sequence and the model is outputted as a detected pattern. Otherwise the empty model is outputted.
4) Steps 1-3 are repeated.

In these experiments, GAIS has used the following parameter values:

- Population size has been 100.
- Chromosome amount has been 4. (This demonstrates the situation that the real amount of interleaved patterns is not known.)
- The mutation rate has been 0.15 (swap mutation rate 0.11 + relocation mutation rate 0.04)
- The empty model has been added to the model set in iteration round 50.

Artificial data has been used to evaluate the performance of the GAIS and the greedy algorithm. Results of the algorithms are easy to interpret and compare when using artificial data. Five English language words are used as real models and four as fake models. Fake models have been chosen to reseble the real models and they have been added to the model set to complicate the detection. Table I shows the models. A set of twenty event sequences have been generated by interleaving three randomly selected real models for each event sequence. These sequences are the basis of the test sequences used in adding and mixing tests, and they are referred to the original event sequences hence.

TABLE I
MODELS USED IN TESTS. EVENT SEQUENCES CONSIST OF REAL MODELS. IN ADDITION TO THE REAL MODELS, THE MODEL SET CONTAINS FAKE MODELS, WHICH COMPLICATE THE DETECTION.

Real models	Fake models
PATTERN	PATH
IMPERFECT	IMPACT
SEQUENTIAL	SEQUENCE
GENETIC	GENERIC
ALGORITHM	

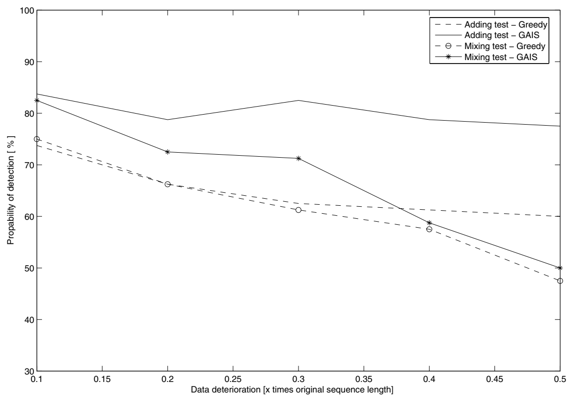


Fig. 5. Comparison of GAIS to the greedy algorithm. Used event sequences were deteriorated by adding events and mixing the time order.

At first, tests for the original event sequences have been run. GAIS is able to detect 86% of patterns and the greedy algorithm 76%.

A. Adding test

In adding tests, new events have been added into the original event sequences. Added events have the same types as events in the models. The addition has been done gradually. Let the length of an original event sequence be l. At first $0.1l$ new events have been added, then $0.2l$, $0.3l$, $0.4l$, and $0.5l$.

Figure 5 shows the results for adding tests. GAIS clearly outperforms the greedy algorithm. GAIS is able to detect about 80% of the patterns regardless of added events.

B. Mixing test

In this test, the time order of the original event sequences has been mixed. Two events have been chosen randomly and their time stamps have been changed. This has been repeated $0.1l$, $0.2l$, $0.3l$, $0.4l$, and $0.5l$ times. Variable l is the length of the original event sequence.

Figure 5 shows the results for mixing tests. GAIS outperforms the greedy algorithm. Up to the level, where time stamps have been changed $0.3l$ times, GAIS is able to detect over 70% of the patterns.

V. Discussion

Section I gives an example of a monitoring system, which supervises a large machine at a factory. Sensors integrated into the machine produce measurements, which are imprecise or even missing. Fault situations are predicted from preprocessed data using sequential model patterns. Changes in physical quantities are slow and they begin weeks before the real fault situation. Slow changes of this kind are hard to discover, especially, if they are related to different fault situations and take place simultaneously. Sequential data from sensors can be analyzed regularly using a method like GAIS. Then faults can be predicted substantially earlier than the real fault situation occurs.

GAIS assumes the existence of models to guide what kind of patterns we wish to detect from the event sequence. In practice, exact models are seldom available. Because GAIS carries out approximate pattern matching, models do not have to be exact but they can have minor errors. Often some events in the model sequence are more significant than the other. In these cases the weighting of events in the models could lead to even better results.

VI. Conclusion

This paper introduced GAIS, a novel method for detecting interleaved patterns from event sequences. A case, where data is of low quality and has errors has been considered. We approached the problem by partitioning the event sequence with the help of a model database and comparing subsequences with the models. A genetic algorithm has been utilized in partitioning. We have implemented GAIS and compared it with the greedy algorithm. Evaluation results show that GAIS clearly outperforms the greedy algorithm, and using GAIS desired sequential patterns can be detected from low quality data.

References

[1] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards sensor database systems," in *Proceedings of the Second International Conference on Mobile Data Management (MDM '01)*. London, UK: Springer-Verlag, 2001, pp. 3–14.

[2] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring streams - a new class of data management applications," in *Proceedings of the 28th International Conference on Very Large Data Bases*. St. Louis, MO, USA: Morgan Kaufmann Publishers, 2002, pp. 215–226.

[3] A. Motro and P. Smets, Eds., *Uncertainty Management in Information Systems: From Needs to Solutions*. Boston, MA, USA: Kluwer Academic Publishers, 1997.

[4] L. Gao and X. S. Wang, "Continually evaluating similarity-based pattern queries on a streaming time series," in *Proceedings of the 2002 ACM SIGMOD International Conference*. New York, NY, USA: ACM Press, 2002, pp. 370–381.

[5] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Commun. ACM*, vol. 20, no. 10, pp. 762–772, 1977.

[6] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[7] C. Bettini, S. Wang, S. Jajodia, and J.-L. Lin, "Discovering frequent event patterns with multiple granularities in time sequences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 2, pp. 222–237, 1998.

[8] B. Chang and S. Halgamuge, "Approximate symbolic pattern matching for protein sequence data," *International Journal of Approximate Reasoning*, vol. 32, no. 2, pp. 171–186, 2003.

[9] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*. Santiago de Chile, Chile: Morgan Kaufmann, 1994, pp. 487–499.

[10] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast subsequence matching in time-series databases," in *Proceedings of the 1994 ACM SIGMOD International Conference*. New York, NY, USA: ACM Press, 1994, pp. 419–429.

[11] L. Harada, "An efficient sliding window algorithm for detection of sequential patterns," in *Proceedings of the Eighth International Conference on Database Systems for Advanced Applications (DASFAA '03)*. IEEE Computer Society, 2003, pp. 73–80.

[12] H. S. Teng, K. Chen, and S. C.-Y. Lu, "Adaptive real-time anomaly detection using inductively generated sequential patterns," *IEEE security and privacy*, vol. 00, pp. 278–284, 1990.

[13] S. Kirkpatrick, J. C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[14] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, vol. 26, no. 1, pp. 1–13, 1996.

[15] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Inspiration for optimization from social insect behaviour," *Nature*, vol. 406, pp. 39–42, 2000.

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of the 1995 IEEE International Conference on Neural Networks*, 1995, pp. 1942–1948 vol.4.

[17] R. Battiti and G. Tecchiolli, "The reactive tabu search," *ORSA Journal on Computing*, vol. 6, no. 2, pp. 126–140, 1994.

[18] M. Srinivas and L. Patnaik, "Genetic algorithms: A survey," *Computer*, vol. 27, no. 6, pp. 17–26, 1994.

[19] J. T. Alander, "Genetic algorithms - an introduction," *Arpakannus*, vol. 1, pp. 19–32, 2001.

[20] R. A. Wagner and M. J. Fischer, "The string-to-string correction problem," *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, 1974.

[21] T. Blickle and L. Thiele, "A comparison of selection schemes used in evolutionary algorithms," *Evolutionary Computation*, vol. 4, no. 4, pp. 361–394, 1996.