Concept change detection and model adaptation for chronicle recognition based diagnosis

Master's Research Report

Submitted by

Qasim Malik

Supervised by

René Quiniou, DREAM (IRISA)

Thomas Guyet, DREAM (IRISA)

Table of Content

1 INTRODUCTION 1

1.1 INTRODUCTION TO ONLINE DIAGNOSIS PROBLEM 1

1.2 DIAGNOSIS BY CHRONICLE RECOGNITION 2

1.2.1 Chronicle representation 3

1.2.2 Chronicle recognition system (CRS) 5

1.3 DIAGNOSIS PROBLEM WITH CONCEPT CHANGE 6

1.4 PURPOSE 7

1.5 CONTRIBUTION 8

2 CONCEPT CHANGE DETECTION 8

2.1 WHAT IS CONCEPT CHANGE IN OUR CASE? 8

2.2 HOW TO DETECT CONCEPT CHANGE? 8

2.2.1 Frequency of chronicle recognition 9

2.2.2 Percentage of completion 9

2.2.3 Killings at each temporal constraint 11

2.2.4 Chronicle vs. sub-chronicle recognitions 12

3 MODEL ADAPTATION 15

4 IMPLEMENTATION 16

4.1 CRS SOFTWARE 16

4.2 DATA GENERATION 17

4.2.1 Generation of chronicle models file 17

4.2.2 Generation of events data file 17

4.3 IMPLEMENTING CHRONICLE VS. SUB-CHRONICLE RECOGNITION TECHNIQUE 18

4.3.1 Sub-chronicle generation 18

4.3.2 Ensuring mutual exclusiveness 19

5 EXPERIMENT 19

5.1 RESULTS 20

5.2 DISCUSSION 22

5.2.1 Idea about the extent of change 22

5.2.2 Change detection 23

6 CONCLUSION 24

REFERENCES 25

APPENDIX 27

SIMULATION RESULTS 27

Abstract

The supervision of dynamic systems, such as telecommunication network, requires a monitoring system for detecting abnormal situations in less amount of time. The monitoring system monitors the generated data streams and issues time stamped alarms. Diagnosing dynamic systems consists of identifying the components failure by analyzing these alarms. We are interested in performing diagnosis using chronicle recognition system (CRS). A chronicle corresponds to a possible abnormal situation and is composed of a set of events (alarms) related by temporal constraints. Such a constraint is a temporal interval whose bounds represent the minimal and maximal delay between the occurrences of two events. The CRS performs the online recognition of chronicles.

Due to streaming nature of event data and the fact that the system is dynamic in nature, changes in the system can cause changes in the pattern in which events are coming. Due to this, our definitions of chronicles may need some changes with the time. When this situation occurs, we say that concept change has been occurred and to overcome it we need to update our knowledge base so that chronicles also adapt to the changes with time. Chronicle adaptation means to modify the temporal constraints by enlarging or narrowing the temporal interval.

The objective of this research work is to detect the concept change. Once it is detected then how to perform chronicle adaptation in order to overcome it. In addition, what kind of history must be managed in order to perform relevant adaptations to chronicles.

In this work, we have proposed three methods for detecting concept change. The two methods are based on two different measures that are ‘percentage of completion’ and ‘killings at each temporal constraint’. In the third method, we introduced the notion of sub-chronicles and used them to detect concept change. Further we have discussed about the kind of history that can guide the adaptation process. We are still working on using this information for performing adaptation. In the end we have evaluated the third method using CRS software by experimenting it on synthetic dataset that we generated.

1 Introduction

1.1 Introduction to online diagnosis problem

In complex dynamic systems such as telecommunication network or patients in intensive care units etc., the data comes out from sensors in huge volume and in continuous flow called data stream. Diagnosing such systems consists of identifying the components failure using observations derived from the streaming data. It requires careful monitoring of the system and reliable identification of abnormal situations indicating failures. The streaming nature of data however makes the task of identifying abnormal situations difficult because due to its huge amount it is not possible to store the whole data and therefore one can have only one look of the data. It gets really difficult for human supervisor to analyze the streaming data and diagnose failures in real time.

The supervision of these dynamic systems therefore requires a monitoring system to identify abnormal situations in less amount of time. The monitoring system keeps track of the system continuously, analyses all situations encountered and suggests decision to be taken in case of abnormality. A typical monitoring system consists of the following three modules [1]:

1. Detection module
2. Diagnosis module
3. Decision module

Figure 1 shows the typical architecture of a monitoring system.

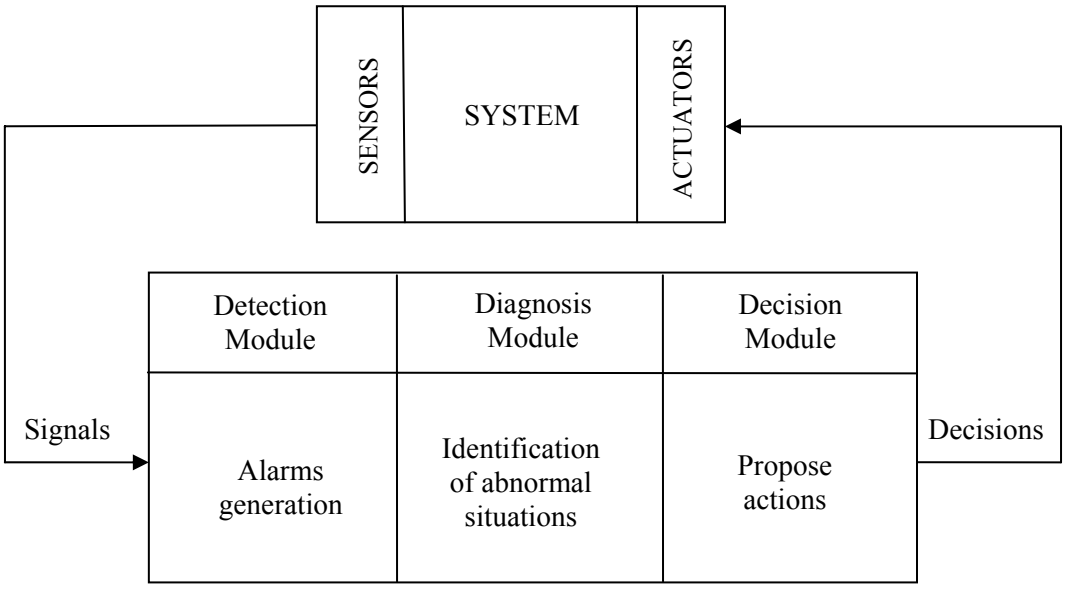


Figure 1 Typical architecture of monitoring system

The detection module continuously monitors the sensor data and decides whether the system is behaving normally. It detects the troubling conditions that occur in the system and issues time stamped

alarms accordingly. Alarms are the messages that indicate troubled conditions and by making it time stamped we also keep the information about the occurrence time of the alarm.

The diagnosis module is responsible for analyzing alarms on the fly in order to identify abnormal situations and thus the diagnosis of corresponding failure. Many approaches have been used for performing the diagnosis task.

Expert systems are widely used in industrial monitoring system for diagnosing dysfunction in particular by IFP (Institut Française du Pétrole) in the Alexip software [14] and by France Telecom for monitoring the Transpac network [15]. They link the set of alarms directly to some characteristic situation, which one needs to identify, in the form of rules. Once these rules are acquired in knowledge base, the task is then to analyze the alarms stream and perform rules validation.

Discrete event models have been used in [18] which describe the system behavior in different modes (normal, degraded, failure). In general, they enable simulating the system step by step, thus predicting the values of observed variables. They can therefore be used directly to detect abnormal situations by mean of comparison between predictions and observations.

Diagnosis can also be performed by recognition of chronicles and has been used in AUSTRAL [17] and GASPAR [16] projects. A chronicle represents a possible abnormal situation that can occur in the system. It consists of events that are interlinked by temporal constraints. The events are basically the alarms generated by the detection module. Figure 2 shows a possible chronicle consisting of four events. The direction corresponds to the order in which events should occur in time. The temporal constraint between *B* and *D* indicates that it can be at least 2 and no more than 9 time units between their occurrences. No temporal constraint between each pair of connected events corresponds to a delay up to infinity [0, +∞ [. The diagnosing task here is to recognize those chronicles on-line.

Once the diagnosis module completes its task, the decision module suggests the corresponding action (if any) to actuator in order to bring back the normal behavior of the system.

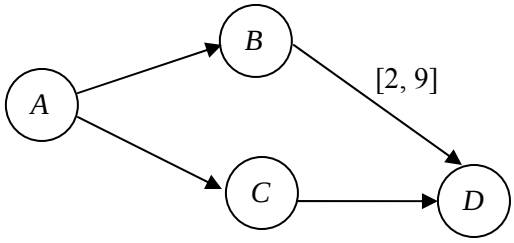


Figure 2 A chronicle

1.2 Diagnosis by chronicle recognition

We are interested in performing diagnosis by chronicle recognition. In our framework, each failure is associated with a chronicle which corresponds to some abnormal situation. The diagnosing task consists of tracking online the occurrences of these chronicles. To perform this task, we require a chronicle knowledge base that contains all the possible chronicles corresponding to different possible

failures and a chronicle recognition system. Acquiring a knowledge base is not a trivial task. In fact the main difficulty in using the chronicle recognition approach is to acquire a chronicle base. Learning techniques have been proposed to remedy this problem and two kinds of techniques have been investigated. One is based on unsupervised learning of chronicles that takes its inspiration from data mining techniques and is presented in [12] while the other is based on supervised learning that relies on the behavioural model of the system and is mainly studied for telecommunication network system in [11] and [13].

The acquired knowledge base is given as input to the chronicle recognition system which analyzes the stream of events and performs online recognition of chronicles. To perform recognition, chronicle recognition system must be presented with chronicles having formal representation.

1.2.1 Chronicle representation

The concept of chronicle was first given by Dousson *et al*. in [3] and they first called it *situation*. We are using the same representation for representing chronicle as used in [2] and [3] which relies on a propositional reified logic formalism in which the environment is described through domain attributes and messages. Their temporal qualifications are formed using different predicates. Below we briefly describe about the domain attributes and messages, the different types of predicates, and an example in which a chronicle describes telecommunication switch problem.

1.2.1.1 Domain attributes and messages

A domain attribute is a couple A:a where A is the attribute name and a is its one of the possible value amongst a set of values called domain. At each point t, a domain attribute can have only one value from its domain. The domain attributes are used to represent the state variables of the system.

A message does not have any domain and is used to represent the transient data.

1.2.1.2 Predicates

A predicate is used in a chronicle to define which events are required for the recognition and which ones should be avoided. In order to be recognized, a chronicle must satisfy all the predicates it contains. There are three kinds of predicates that can be used in defining a chronicle. All these predicates rely on attributes and/or messages. Below we take a look at them one by one.

1. To represent persistency of the value of a domain attribute A over an interval $[t_1, t_2]$ without knowing when this value was reached, a predicate *hold* is used. This predicate can only be applied on domain attributes.

$$hold\ (A{:}\ a, (t_1, t_2))$$

2. An event pattern corresponds to a change of the value of a domain attribute or an occurrence of some message. An event is a time stamped instance of an event pattern and it has no duration. It is expressed by the predicate *event*.

$$event\ (A: (a_1, a_2), t)$$

$$event\ (M, t)$$

The first *event* predicate corresponds to a change in the value of domain attribute A from a_1 to a_2 at time t while the second corresponds to an occurrence of the message M at time t.

3. A forbidden event is defined using the *noevent* predicate. It can be used for both domain attributes and messages.

$$noevent(A: (a_1, a_2), (t_1, t_2))$$

$$noevent(M, (t_1, t_2))$$

The first *noevent* predicate forbids the change in the value of domain attribute A from a_1 to a_2 while the second forbids the occurrence of a message M within time interval $[t_1, t_2]$.

1.2.1.3 An example of chronicle representation

A chronicle is composed of the following three parts:

- a set of predicates
- a set of temporal constraints which relate these predicates, and
- a set of actions to be taken when the chronicle is recognized

To show the chronicle representation, we take an example from [2] in which a chronicle describes a telecommunication switch problem: the transmission link with the equipment is lost at time instant t_1 and is then recovered after rebooting at time t_2 which needs up to 3 minutes. After that, all three components of the switch send a status message at time t_3, t_4, and t_5 within 1 or 2 minutes. This chronicle is described using the above representation in Figure 3 and the associated temporal graph is shown in Figure 4. The character '*' is used when we do not care about the corresponding value. It could be any value from the associated domain.

```
chronicle SwitchProblem {
    //forthcoming events
    event(Transmission:(on,off),t1)
    event(Transmission:(off,on),t2)
    event(Component1:(*,ok),t3)
    event(Component2:(*,ok),t4)
    event(Component3:(*,ok),t5)
    //temporal constraints with time specified in seconds
    t1 < t2 < t3
    t2 < t4
    t2 < t5
    (t2 - t1) in [0,180]
    (t3 – t2) in [60,120]
    (t4 – t2) in [60,120]
    (t5 – t2) in [60,120]
     when recognized {
      print 'switch problem detection'
    }
}
```

Figure 3 Chronicle for the identification of telecommunication switch problem

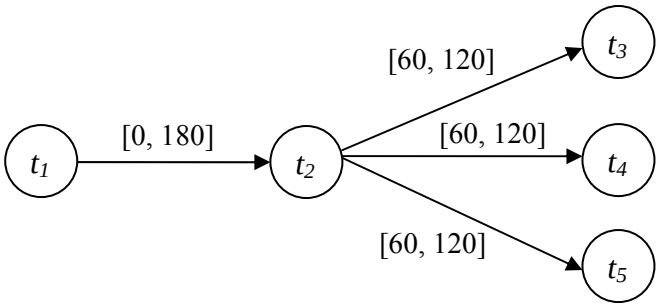


Figure 4 Temporal constraint graph for the chronicle of Figure 3

1.2.2 Chronicle recognition system (CRS)

Given some knowledge base of chronicles, the job of the CRS is to analyze the input stream of events and to recognize on the fly the sequences that match any of the chronicles in knowledge base. The chronicle recognition algorithm was first proposed by Dousson *et al*. in [1] and some improvements have been proposed to this algorithm in [4] and [5]. This algorithm is implemented in the CRS software [6] developed by Christophe Dousson. We have used CRS software for chronicle recognition in this research work. It uses the same representation of chronicles that is described in the previous section. Below we describes about its working.

A specific event in a chronicle serves as a trigger event or starting event. When this event is detected in the stream, a partial instance of that chronicle is created. A partial instance contains events that match with a subset of chronicle events and satisfies the temporal constraints in between. When complete matching is found, the chronicle is recognized.

Once the partial instance gets created, the stream of events is analyzed and compared with the chronicle. The recognition process is done in a way to ensure that the time constraints are never violated in all the partial instances. If a time constraint is violated, the related partial instance gets killed. There are only two ways for killing a partial instance:

1. an event occurs during a time interval in which it is forbidden (*noevent* predicate)
2. an event does not occur within the desired time interval and thus violates the temporal constraint

On the other hand, a partial instance leads to creation of another one if new event satisfying the temporal constraint arrives. This is because whenever a new event satisfies the specification of a chronicle and needs to be integrated in a partial instance, the original partial instance is duplicated and the event is integrated only in the copy of it. This duplication is necessary and guarantees the recognition of chronicle as often as it may arise. To have a clear idea let us take an example of a simple chronicle model shown in Figure 5.

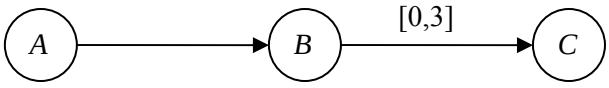


Figure 5 A simple chronicle model

Let say we receive following stream of events:

(*A*, 0)

(*B*, 3)

(*B*, 8)

(*C*, 11)

Table 1 shows all the partial instances that would be created and killed during the recognition process. Notice that had there been no duplication of partial instance **{**(*A*, 0)**}** at time 3, we may not be able to recognize the chronicle at time 11. Also note that the partial instance **{**(*A*, 0)**}** will never be killed because its associated temporal constraint allows it to wait for event *B* up to infinity and therefore will never get violated. So if in future we receive events *B* and *C* within desired time, we will get another recognition. In general, an event may be shared by many chronicles and the CRS will be able to manage all the concurrent instances.

Table 1 Partial instance creation during recognition process

Time	Partial Instances
0	**{**(*A*, 0)**}**
3	**{**(*A*, 0)**}**, **{**(*A*, 0), (*B*, 3)**}**
6	**{**(*A*, 0)**}**
8	**{**(*A*, 0)**}**, **{**(*A*, 0), (*B*, 8)**}**
11	**{**(*A*,0)**}**, **{**(*A*, 0), (*B*, 8)**}** (recognition)
12	**{**(*A*,0)**}**

The recognition performed by CRS is exhaustive i.e. all the occurrences of defined chronicles are recognized by the system. The CRS is incremental in nature as each event is integrated as soon as it occurs and it is performed in single reading of the input stream. Also this system is predictive in the sense that it predicts forthcoming events that are relevant to partial chronicle instances currently under development.

1.3 Diagnosis problem with concept change

The systems, we are considering, are dynamic in nature. They can undergo through various changes and evolve with time. Their evolution could be due to changes in their own dynamics or under the impact of external actions. For example, in case of telecommunication system, components may be changed in a system; in case of patients in intensive care unit, drugs can be given to them, etc. Furthermore, these systems produce data in the form of stream. Data stream represents input data that is produced continuously in huge amount. Data streams differ from the conventional stored data in several ways:

- The data elements in the stream arrive online.
- Data streams are potentially unbounded in size. The whole data can not be processed at once. So we have to process the data in the form of batches or use sliding window.
- Once an element from a data stream has been processed it cannot be retrieved unless it is explicitly stored in memory, which typically is small relative to the size of the data streams.
- The data's underlying concept may change over time. Concept change, according to Tsymbal [7], refers to more or less radical changes in the target concept induced by changes in the hidden context not known explicitly. As our system is dynamic in nature and produces data in the form of stream, so when it will exhibit some change it will also cause the change in the pattern in which data is produced.

In our case the stream is composed of events. The diagnosis module analyses the events in order to recognize chronicles. The chronicles are our target concepts. Their recognition essentially depends on the temporal gap between events and temporal order in which events are coming. The temporal gap and ordering may get affected due to changes in the system not known explicitly. For example, addition of new components into the system may cause the change in the pattern in which events are issued. Thus as a result of changes in the system, we may not be able to recognize the chronicles and changes may be needed in out target concept so as to perform diagnosis correctly. Take the example of the chronicle of Figure 4, for instance, that is used for detection of switch problem. Due to changes in the system, if the rebooting process begins to take more than 3 minutes then we will not be able to recognize this chronicle.

So when a situation occurs where there is a need for changing the definition of at least one chronicle, we will say that concept change has occurred. This causes a need to update our knowledge base so that chronicles also adapt to the changes with time.

The concept change hinders the performance of the diagnosis module and it may not be able to recognize all the occurrences of chronicles. To avoid this, we have to first detect the concept change and then overcome it. For overcoming the concept change, we can either build a new model of chronicles or perform necessary adaptations to it. As we are performing online diagnosis and dealing with data stream, so we can not afford to build new models as this process will consume a lot of time. The viable solution for this case is to perform necessary adaptations to the existing chronicle models. Chronicle adaptation means, for instance, to add or delete an event, or to modify the temporal constraints by enlarging or narrowing the temporal interval.

1.4 Purpose

The problem of concept change has been discussed in the literature and the solutions have been proposed to overcome this change by performing knowledge base update. The problem of concept change is well defined and related work is presented in [7]. But the solutions presented in [8], [9], and

[10] for the concept change problem are in the context of classification learning in which time information is not taken into account.

The purpose of this research work is to first detect a situation where there is a concept change and thus there is a need for chronicle base adaptation. Once the change is detected, then how to actually perform chronicle adaptation on the fly. In addition, what kind of history must be managed in order to perform relevant adaptations to the chronicles representing abnormal situations, given that the volume of the data on the stream is such that it is impossible to record the whole data.

1.5 Contribution

In this research work, we have proposed three methods for concept change detection. The first two methods are based on heuristic measures and they use these measures to detect the concept change. In the third method a chronicle is divided into sub-chronicles and their recognitions help in detecting the concept change. Once the change detection is done, then the next step is to perform chronicle model adaptation. We have therefore introduced an information that can guide us in performing adaptation. The process of using this information for adaptation is still in progress.

Due to some limitations that will be described later, we did not implement the first two methods. The third method was implemented in which we used CRS software for recognizing chronicles and sub-chronicles and finally experimented it on synthetic dataset.

2 Concept change detection

In this section, we formally define what we mean by concept change in our framework and introduce three ideas proposed for detecting the concept change.

2.1 What is concept change in our case?

During the process of online chronicle recognition, when a situation occurs where there is a need for changing the definition of at least one chronicle, we will say that concept change has occurred. Although the concept change could prompt either the addition or deletion of an event in a chronicle or the modification of temporal constraints by enlarging or narrowing the temporal interval but we only limited our work to the case in which concept change can only enforce the modification in temporal constraints.

2.2 How to detect concept change?

As the event data is of streaming nature, so we can not store the whole data due to limited storage capacity. We have to process the events stream in the form batches or using sliding windows. As you will be detailed later, all the proposed ideas require maintaining a recognition profile for each of the chronicles. This is not possible for the case of sliding window. We were therefore impelled to process

data steam in the form of batches. Each batch consists of a fixed number of events. The size of the batch depends on the resources we have in term of memory. It should be large enough so that we can have a sufficient number of chronicle recognitions within a single batch.

Now we look at the ideas proposed for detecting concept change.

2.2.1 Frequency of chronicle recognition

From each batch, we can obtain the frequency of recognition for each chronicle. The initial idea that comes to mind for detecting change is that for a given chronicle if its frequency is continuously very low for some batches and is also well below than some frequency observed earlier then we may conclude that the change has been occurred in that chronicle. But it is not sufficient for conclusion. The low or zero frequency could be due to the fact that the abnormal situation to which this chronicle corresponds is not occurring frequently in the system. Hence there are two possibilities for low frequency of chronicle recognition: either a change has been occurred and the chronicle needs to be updated or the failure associated with this chronicle is not occurring. We are interested in the former one. In order to separate these two cases, we are introducing two heuristic measures that are "percentage of completion" and "killings at each temporal constraint".

2.2.2 Percentage of completion

Once a partial instance of the chronicle is created, the sequence of events coming in stream are analyzed and compared with the chronicle. If the sequence of events satisfies the chronicle specification then the new updated partial instances are created, otherwise it is killed. This goes on until the partial instances get killed or recognized completely.

Once an event does not satisfy some temporal constraint, what we can do that instead of killing the partial instance we can ignore it and go on with the recognition process assuming that the event arrival may have been delayed due to change. This will give us the number of temporal constraints that a partial instance of some chronicle has not satisfied. Using this we can find the percentage of completion for some partial instance as:

Percentage of completion = No of temporal constraints satisfied / Total no of constraints

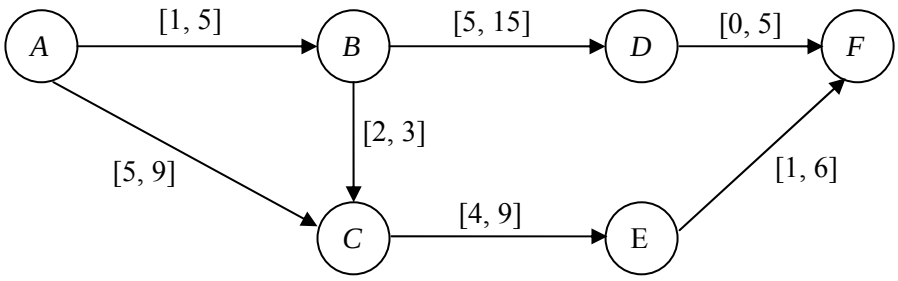


Figure 6 A chronicle

In order to explain it further, consider a chronicle of Figure 6. Let say we have received the following sequence of time stamped events:

(*A*, 36)

(*B*, 39)

(*C*, 43)

(*E*, 47)

(*D*, 48)

(*F*, 49)

In the above sequence of events, the temporal constraint between event *B* and *C* gets violated because we did not encounter event *C* between time unit 41 and 42. But the remaining temporal constraints are satisfied. So the percentage of completion in this case is 6/7.

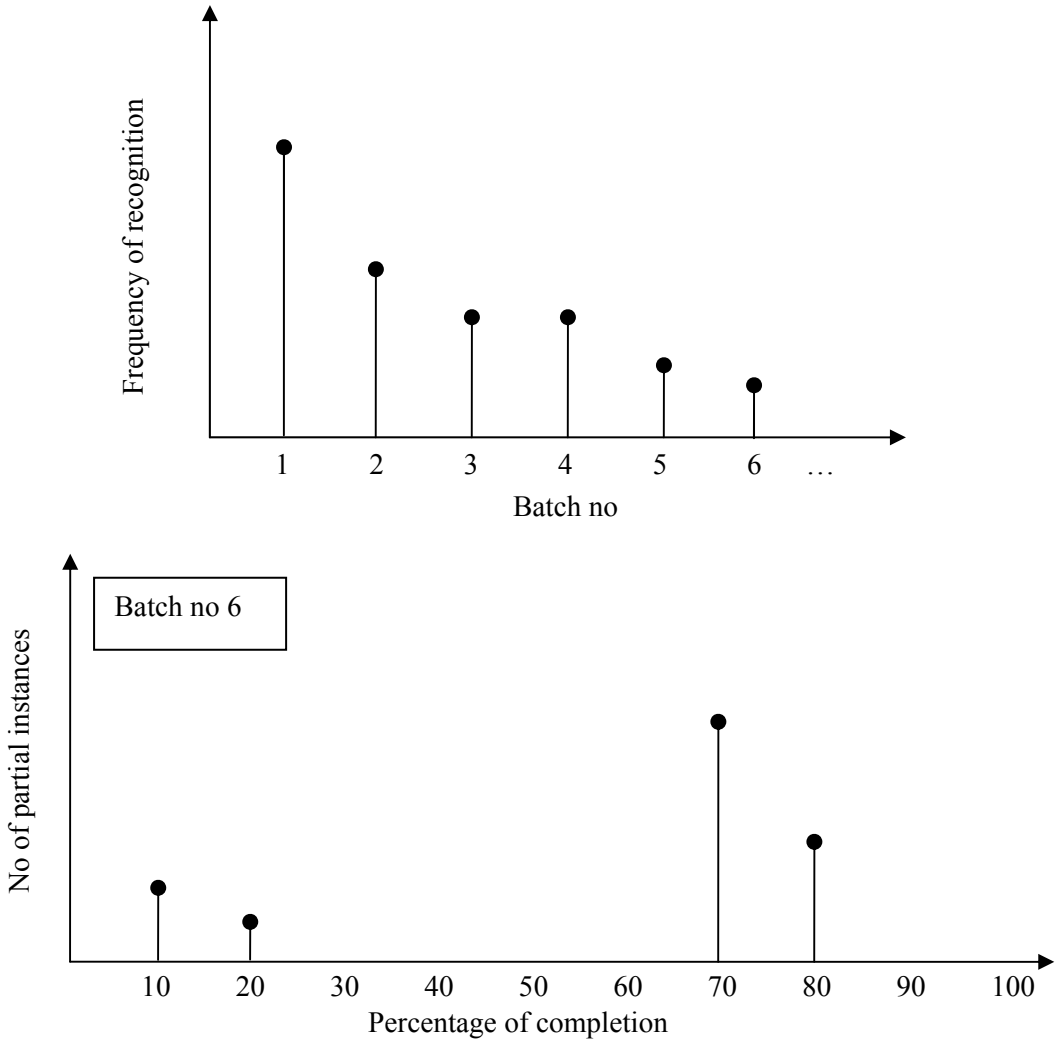


Figure 7 Batch wise profile needed to be maintained for each chronicle. The upper figure illustrates the frequency of recognition of a chronicle in successive batches. The lower figure illustrates the number of partial instances that each bin contains for 6th batch

We can divide the whole range of percentage of completion into ten bins where each bin is of size equal to 10. All the partial instances with percentage of completion from 0-10% will be put in first bin

and so on. If the number of partial instances of some chronicle with percentage of completion, let say >70%, is large and the frequency of its recognition is low then we can say that a change has occurred in that chronicle. For this, we maintain, for each chronicle, the batch wise profile for frequency of recognition and number of partial instances in each bin of percentage of completion as shown in Figure 7.

Shortcomings

This approach has many shortcomings which are enumerated below:

1. Normally each partial instance gets killed when it does not satisfy the chronicle specification and thus reduces the instances to look out for. But in this approach, each partial instance will be kept for at least as much time as it is required to get it recognized completely. This will require more resources in terms of memory and processing.
2. By not satisfying the chronicle specification, we mean that the upper limit of temporal constraint has been reached and the required event was not detected. We are thus assuming that if there is a change, it is due to increase in upper limit of temporal constraint. It is also possible that we have not detected the required event because it may have occurred even before the lower limit of temporal constraint.

   Consider Figure 6 again, let say this time we have received following sequence of events,

   (*A*, 34)

   (*C*, 36)

   (*B*, 39)

   (*E*, 41)

   (*D*, 45)

   (*F*, 47)

   In this case, the event *C* occurred before its lower time limit. Once *C* had occurred, the event *E* occurred within its specified temporal constraint (i.e. 5 is in [4, 9]). But according to our approach, we will wait up to 9 time units for event C. Due to this we will also miss the event *E*.
3. The other shortcoming is the assumption upon which this measure is based on that the events are not related with each other and are independent which is a rarity. During the process of recognition if some event does not occur, then we can only ignore it and go on with the recognition process if the following event is independent of it.

Due to lot of shortcomings it offers, we introduced another measure that is “killings at each temporal constraint”.

2.2.3 Killings at each temporal constraint

When a chronicle is in the process of recognition, a lot many partial instances are created and killed. These partial instances are killed when some temporal constraint is violated. We are interested in

finding a temporal constraint that was responsible for not letting the chronicle recognized. The killed partial instances may be sub-instance of some other partial instances. A sub-partial instance contains events that match with a subset of events in its super partial instance. Such a sub-partial instance may have been killed due to two reasons:

1. It was waiting for the same event once again that has already been incorporated in its super-instance.
2. It was waiting for some other event that is not present in its super-instance. In that case it will be killed along with its super-instance because super-instance would also have been waiting for that event.

A partial instance that is not a sub-instance of any other partial instance will be the largest partial instance that has been recognized. If this partial instance is killed, this means that the chronicle, currently under recognition process, is killed. So if we can find that why largest partial instance was killed, we can know that why a chronicle was not recognized.

We are interested in finding the number of largest partial instances killed at each temporal constraint of each chronicle. If the frequency of some chronicle is low and frequency of killing of largest partial instances at some temporal constraint of this chronicle has increased, this may mean that change has occurred due to this temporal constraint. This measure has the advantage that it tells about where exactly the change has occurred.

We were interested in implementing this measure for detecting change, but there was no functionality available in CRS software that could provide us the temporal constraint responsible for killing the largest partial instance.

2.2.4 Chronicle vs. sub-chronicle recognitions

A chronicle can be thought of as a graph in which vertices represent events and edges represent the temporal constraints between event couples. A graph is said to be connected if it is possible to reach any vertex starting from any other vertex by traversing edges in some direction (i.e. not necessarily in the direction they point). In the same way we can define connected and disconnected chronicles.

In this approach, we divide a chronicle into sub-chronicles by removing the temporal constraints one by one. Then we put them along with original chronicles in our knowledge base. For example Figure 9 shows different possible sub-chronicles for the chronicle of Figure 8. Note that there will be as many sub-chronicles as the number of temporal constraints. Now when we remove a temporal constraint and obtain a sub-chronicle, there are two possibilities;

1. The sub-chronicle remains connected
2. The sub-chronicle may get divided into two parts.

The third sub-chronicle in Figure 9 shows the 2nd possibility. Now we deal these two cases one by one.

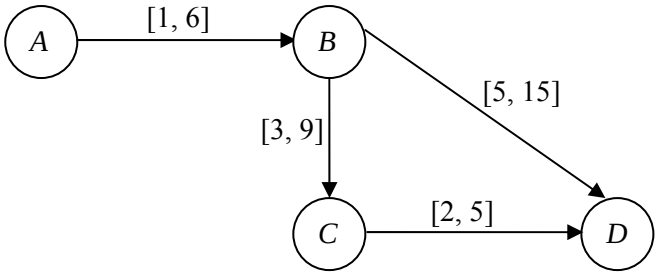
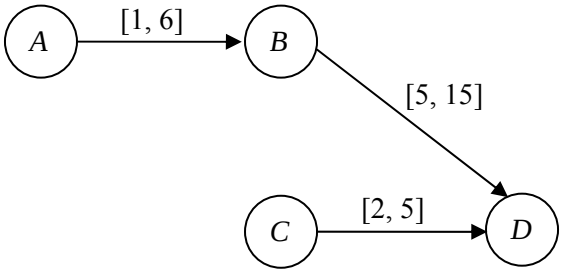
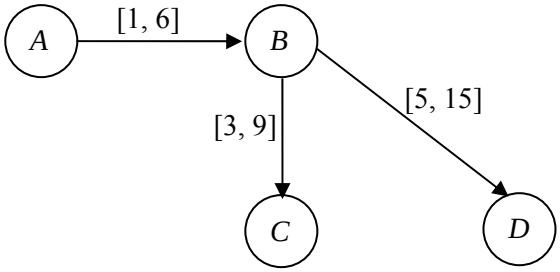


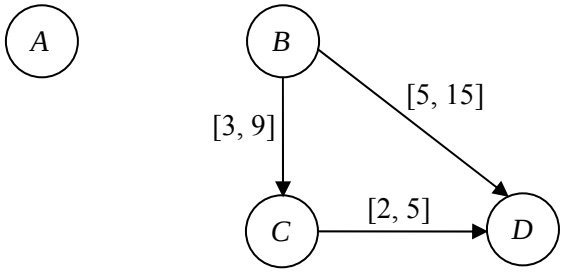
Figure 8 A chronicle



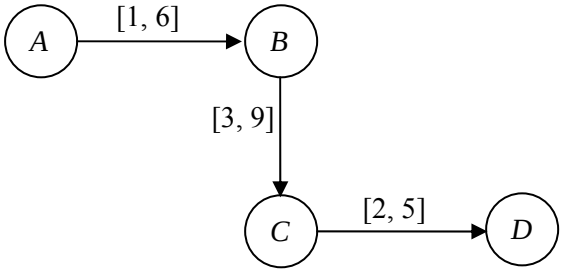
a) - Sub-chronicle 1



b)- Sub-chronicle 2



c) - Sub-chronicle 3



d)- Sub-chronicle 4

Figure 9 Sub-chronicles for chronicle of Figure 8

2.2.4.1 First case

In this case we will have a connected sub-chronicle which implies that each event of a sub-chronicle will be linked with rest of the sub-chronicle. If a sub-chronicle is recognized and original chronicle is not recognized then we may conclude that the change has occurred and is due to the temporal constraint whose removal caused the formation of the sub-chronicle.

2.2.4.2 Second case

In this case the sub-chronicle will not remain connected and we will have its two sub parts. This case requires some attention as we have to specify some sort of connection between two sub-parts of disconnected sub-chronicle. For that there are two possibilities:

1. Make the temporal constraint equal to infinity
2. Expand the already present temporal constraint

If we make the temporal constraint equal to infinity then we will get unnecessary recognitions of sub-chronicles. To have a clear idea let's say we connect the event *A* and *B* and make the temporal constraint between them equal to infinity as shown in Figure 10.

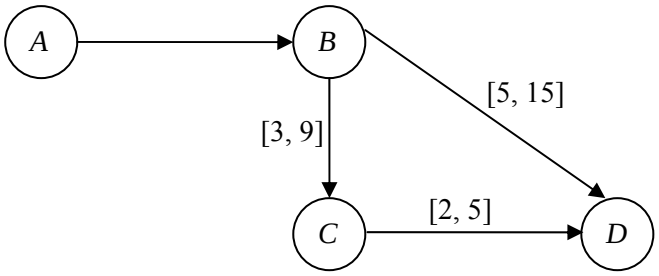


Figure 10 Connecting the two parts of sub-chronicle through an infinite temporal constraint

Whenever we will receive an event *A* in the data stream, a partial instance will be created. This partial instance will never be killed because the associated temporal constraint is infinity and it will never get violated. Whenever we will receive events *B*, *C* and *D* within specified temporal constraints we will have a recognition of sub-chronicle. We will get as many recognitions as number of times the event *A* would have been encountered in the data stream. This does not seem to be viable solution.

The second possibility is that instead of removing the temporal constraint, we can expand it. The question then arises that how much to expand it. This depends on the time interval of the temporal constraint and the physical situation to which this delay corresponds to. For example in the chronicle of Figure 4, the temporal constraint between t_1 and t_2 is from 0 to 3 minutes. This actually corresponds to rebooting time. If we can have some idea about change in rebooting time due to occurrence of changes in the system, then this may help in expanding the temporal constraint.

Now once connected, this case becomes similar to the first one and same idea can be applied for change detection.

2.2.4.3 Criteria for detecting concept change

As the original chronicle differs from its sub chronicles by only one temporal constraint, so the probability of occurrence of any sub chronicle without the occurrence of original chronicle, when the change is not there, is very low. We also insure the mutual exclusiveness between chronicle and its sub-chronicles recognition i.e. if a sub-chronicle gets recognized this implies that the original chronicle has not been recognized and vice versa. By combining the above two, we can conclude that if a sub-chronicle is recognized then there is high probability that the original chronicle has not been recognized due to the change.

2.2.4.4 Limitations

This proposed approach has the following limitations:

1. The change in the chronicle could only occur in one of the temporal constraint at a time. If change occurs in more than one temporal constraint, then we will not get recognition of any sub-chronicle and thus will not be able to detect the change.
2. For change detection, it requires that the chronicles must not have significant similarity with each other. By significant similarity, we mean that they must not have exactly similar organization of events with difference in some of the temporal constraints. Consider the case of two chronicles, for example, that differs in only one of the temporal constraint. In that case if change in one of the chronicle is due to that very temporal constraint then we may get recognition of both the sub-chronicles and may make false decision about the change detection for other chronicle.

3 Model adaptation

The detection of concept change implies that at least one of the chronicles in the knowledge base needs some modification. Unless the modifications are made, we may not be able to recognize chronicle when it will arrive. This will harm our diagnosis task as we may not be able to identify the corresponding abnormal situation. So once the concept change is detected, it must be overcome by performing chronicle adaptation. In this research work, we considered that concept change can only cause modifications in temporal constraints. So, the chronicle adaptation here refers to modifications in temporal constraints either by narrowing or enlarging them. To do that, we need to maintain some kind of history that can guide us in performing adaptation.

For chronicle to be recognized, all of its events should arrive within the time interval specified by their corresponding temporal constraint. Once a chronicle is recognized we can find, for each temporal constraint, the difference between timestamps of associated events couple. This will give a value that would lie within time interval specified by this temporal constraint. For example in the temporal constraint shown in Figure 11 if event *C* arrives at t_1 and event *D* arrives t_2, and if the chronicle to which this temporal constraint belongs to has been recognized, then $t_2 - t_1$ must lie in the time interval [2, 7]. We are interested in finding this value $t_2 - t_1$ for each temporal constraint of recognized chronicle. We call this value as 'time delay'.

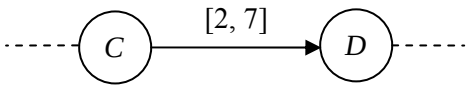


Figure 11 A temporal constraint between an event couple

The time delay can be recorded for each temporal constraint, whenever we get the recognition of some chronicle. We are interested in using this key information for guiding the adaptation process. For this, we need to maintain the history of time delay so that shifting trend of it can help us to decide the direction ($+\infty$ or $-\infty$) in which the time interval should move. Further measures such as mean, minimum, and maximum value of time delay for each batch can help us in perform modifications to

the temporal constraint. We are still working to device a method which uses the time delay information for chronicle adaptation.

Once we determine the new time interval of a temporal constraint, it must be updated in the corresponding chronicle present in knowledge base.

To conclude the description of our propositions, the Figure 12 shows the overall process of concept change detection and adaptation that we followed in our work.

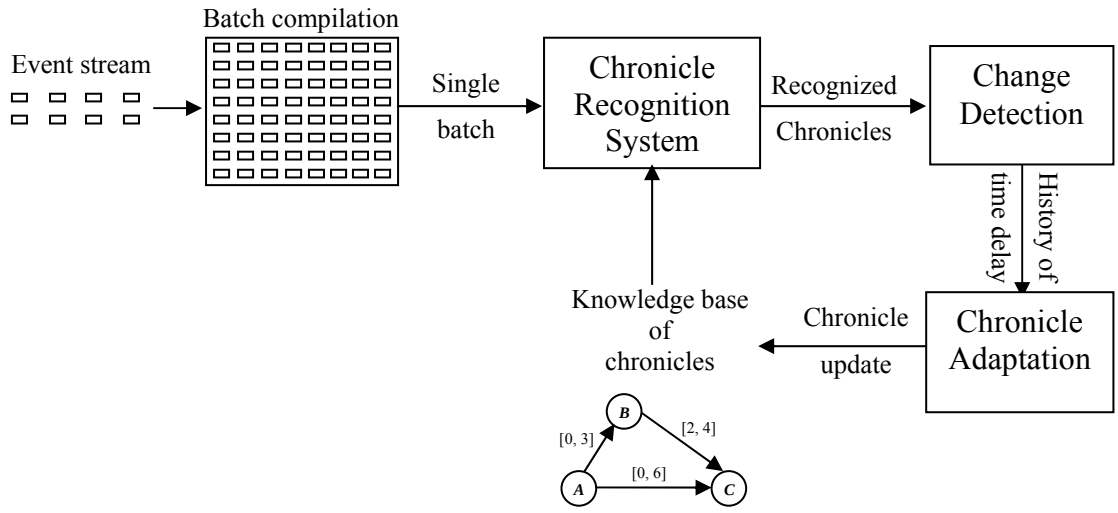


Figure 12 Overall picture of the whole process

4 Implementation

This section explains the implementation process that we followed during our work. First we briefly introduce the CRS software used for chronicle recognition, then the data generation process and finally the process involved in the implementation of change detection technique.

4.1 CRS Software

In our work we used chronicle recognition software, called CRS (Chronicle Recognition System), developed by C. Dousson [6]. The CRS software is in charge of analyzing the input stream of events and of identifying, on the fly, any pattern matching a situation described by a chronicle. In order to carry out this, it requires two files as input to it. First the file that contains chronicle models that we want to recognize. The models must be represented using the representation provided in section 1.2.1. More detail about chronicle language can be found in [6]. Second it requires file containing events issued by monitoring system. These events will be analyzed by CRS software to recognize provided chronicles. The CRS software is written in Java language and provides many useful classes for users to play with. The documentation of complete Java application programming interface (API) that CRS offers can be found in [6].

4.2 Data generation

In this research work, we needed events dataset containing event data before and after the occurrence of change in the system. We had a dataset containing events data related to electrocardiogram but there was no occurrence of change within it. So we had to make our own synthetic dataset. As we are using CRS software for performing chronicle recognition, so the data must be represented as required by CRS. The CRS requires two files as input to it. Below we describe about the generation of these files. The program for data generation was written in MATLAB.

4.2.1 Generation of chronicle models file

To generate chronicle models, we need to first define the chronicle and then write it in the file using desired representation. The definition of chronicles includes the following elements:

- Number of events
- Organization of events within chronicles
- Temporal constraints between each pair of connected events

First we define the different type of events that will constitute the chronicles and then limit the number of events that each chronicles can have by some maximum and minimum number. For each chronicle, number of events is randomly chosen within these limits and their types are chosen randomly amongst the defined types.

For the sake of simplicity, we organized the events within each chronicle in a sequential manner as shown in Figure 13.

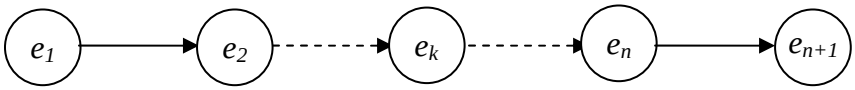


Figure 13 Sequential chronicle

In order to specify the temporal constraint between each pair of connected events, we define the maximum and minimum limits for choosing mean and variation. Both means, μ, and variations, σ, are randomly chosen for each temporal constraint within specified limits. The temporal constraint then takes the form [μ- σ, μ+σ].

Once the chronicle is defined, it is written in the file using desired representation.

4.2.2 Generation of events data file

For generating the events dataset, we need to specify the following:

- Total number of events
- Total number of chronicles to be inserted
- How to introduce the change

The types of events to be included in the dataset were amongst the events defined in the chronicle models file. These events were time stamped and were inserted in the file using uniform random

distribution. Within these events we inserted the specified number of chronicles which were chosen from chronicle base using uniform random distribution.

In order to simulate the data stream, we processed the event data in the form of batches. CRS was provided with fixed size batch of events at one time. To introduce the change within event data, we randomly specified a batch number, for few randomly chosen chronicles, from where the change will begin to occur. The change in chronicle was introduced by expanding the temporal interval of any randomly chosen temporal constraint.

4.3 Implementing chronicle vs. sub-chronicle recognition technique

We implemented this technique for detecting concept change. The implementation required the following steps:

1. Generate the sub-chronicles for each of the chronicle present in knowledge base by removing the temporal constraint one at a time. Each removal of temporal constraint from a chronicle led to a creation of a sub-chronicle. So for each chronicle, there would be as many sub-chronicles as number of temporal constraints.
2. Ensure the condition of mutual exclusiveness for recognition between chronicle and each of its sub-chronicles.
3. Insert these sub-chronicles into the knowledge base along with original chronicle.

4.3.1 Sub-chronicle generation

To generate sub-chronicles, we come across with two cases when we remove the temporal constraint from a chronicle. Either the sub-chronicle will remain connected or it will become disconnected. The former case is easy to handle as we just have to delete the temporal constraint between two events. Figure 14 shows this case for the chronicle of Figure 8 in which sub-chronicle was obtained by removing of the temporal constraint between event *B* and *C*.

The other case is the one that we encounter in our simulation as all the chronicles are sequential and removal of temporal constraint always leaves the sub-chronicle disconnected. In that case we remain them connected and just expand the time interval depending on the mean and variation of it.

```
chronicle chron1[](){
      event(A, R0)
      event(B, R1)
      event(C, R2)
      event(D, R3)
      (R1 - R0) in [1, 6]
      (R2 – R1) in [6, 9]
      (R3 – R2) in  [2, 5]
      (R3 – R1) in  [5, 15]
}
```

```
chronicle subchron1_1[](){
      event(A, R0)
      event(B, R1)
      event(C, R2)
      event(D, R3)
      (R1 - R0) in [1, 6]
      (R3 – R2) in  [2, 5]
      (R3 – R1) in  [5, 15]
}
```

Figure 14 Original chronicle (left) and its sub-chronicle (right) obtained after removing the temporal constraint between event *B* and *C*

4.3.2 Ensuring mutual exclusiveness

If a chronicle gets recognized, its all of the sub-chronicles will also be recognized. This is in contradiction to our criteria for change detection. To avoid this, we needed to ensure the mutual exclusiveness between chronicle and each of its sub-chronicles recognition. For mutual exclusiveness we used the *noevent* predicate in each sub-chronicle. This can be seen in Figure 15.

```
chronicle chron1[](){
      event(A, R0)
      event(B, R1)
      event(C, R2)
      event(D, R3)
      (R1 - R0) in [1, 6]
      (R2 - R2) in [3, 9]
      (R3 – R2) in  [2, 5]
      (R3 – R1) in  [5, 15]
}
```

```
chronicle subchron1_1[](){
      event(A, R0)
      event(B, R1)
      noevent(B,(R0+1,R0+7))
      event(C, R2)
      event(D, R3)
      (R1 - R0) in [1, 6]
      (R3 – R2) in  [2, 5]
      (R3 – R1) in  [5, 15]
}
```

Figure 15 Introducing mutual exclusiveness between original chronicle (left) and its sub-chronicle (right) using *noevent* predicate.

Now if event B arrives within its desired time and chronicle gets recognized, the sub-chronicle would not get recognized due to violation of *noevent* predicate. *B* has to arrive before or after its desired time interval in order to ensure the recognition of sub-chronicle. In that case the chronicle will not be recognized.

5 Experiment

To test the solution proposed in section 2.2.4, we performed an experiment that was carried out using following set of parameters for data set generation:

Number of distinct chronicles = 12

Number of distinct events = 14

Number of events in each chronicle ranges from 6-11

Number of chronicles occurrences that event file will contain = 4819

Number of events in the file = 127500

Batch size = 1500

No of batches = 127500 / 1500 = 85

Table 2 shows the randomly generated parameters used for introducing change in the dataset. For each chronicle, it first shows whether the change was introduced in it and if it was then in which temporal constraint and it started to occur from which batch.

Table 2 Parameters required for introducing change

Chronicle no.	Change to be introduced?	Temporal constraint no.	Batch no.
1	Yes	5	46
2	No	-	-
3	Yes	7	39
4	Yes	1	47
5	No	-	-
6	No	-	-
7	No	-	-
8	Yes	3	39
9	Yes	6	39
10	Yes	4	35
11	Yes	2	55
12	Yes	2	56

5.1 Results

Following are the graphs that we obtained for the first three chronicles. The graphs for rest of the chronicles are provided in appendix. In each of the graph

- the x-axis corresponds to batch number and
- the y-axis corresponds to frequency of recognitions

For the chronicles in which change was introduced, the graph shows the frequency of recognitions of chronicle, its sub-chronicle and the difference between the frequencies. The appearance of bold red line in the 'Difference' graph indicates the detection of change in the chronicle. For the chronicles in which no change was introduced, we only have graph for frequency of recognition of chronicle.

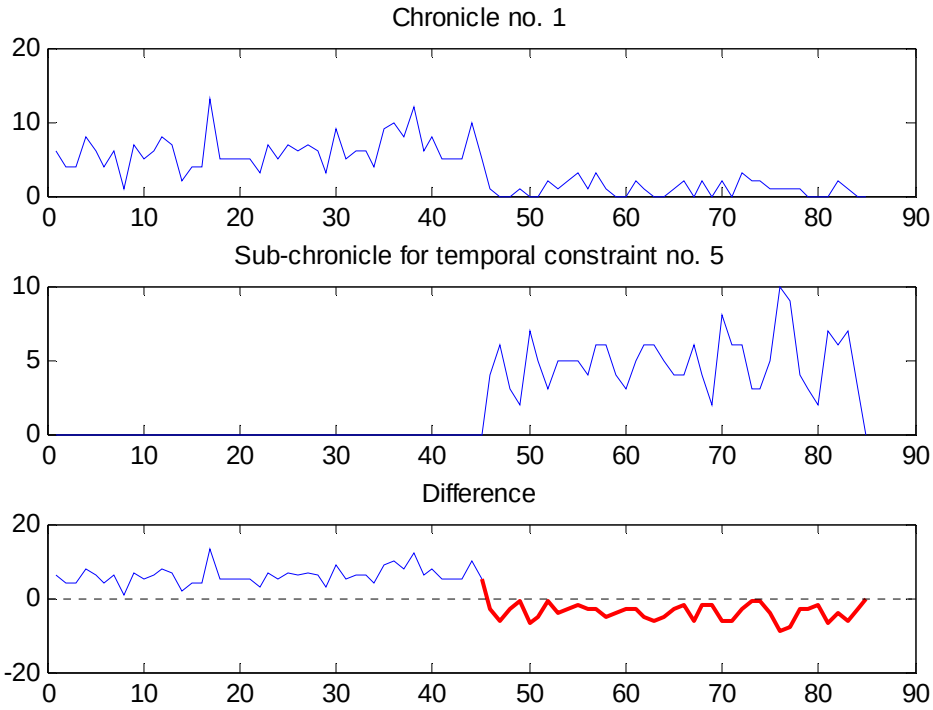


Figure 16 Graphs showing change detection in first chronicle

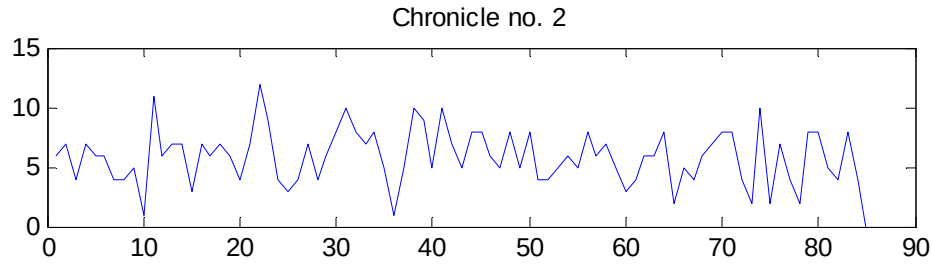


Figure 17 Graph showing no detection of change in second chronicle

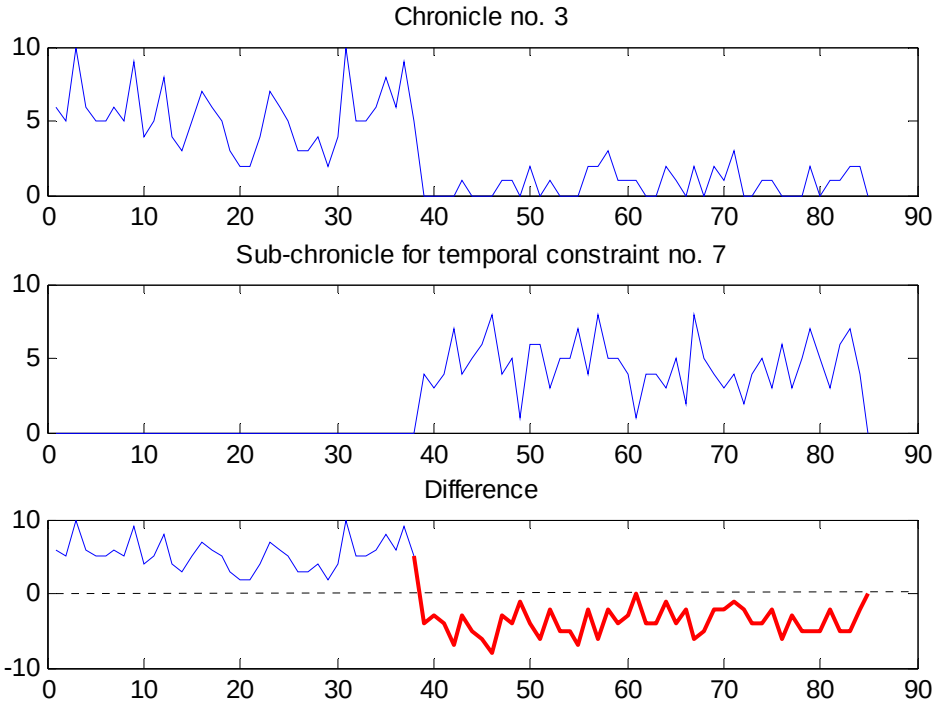


Figure 18 Graphs showing change detection in third chronicle

5.2 Discussion

The above graphs show that the change has been detected in the chronicles in which it was introduced and it has been detected as soon as it occurred. The results satisfy all the parameters mentioned in Table 2.

These results look quite ideal, but there were few important issues that were dealt ideally:

- We already had the idea about the extent of change
- Decision of change detection was made on a single recognition of sub-chronicle

We now talk about them.

5.2.1 Idea about the extent of change

In our case all the sub-chronicles got divided into two parts when they were formed by removing one of the temporal constraints of the chronicles. These parts were then connected by another temporal constraint that was relaxed version of previously removed temporal constraint. In our simulation, we assumed that the concept change will delay the occurrence of events in such a way that it will still fall within the relaxed temporal constraint. In other words, we already had the idea that how much change can occur in each of the temporal constraint.

5.2.2 Change detection

We decide about the detection of change once we encounter a single recognition of sub-chronicle. This works due to the assumption we described in section 2.2.4.4 according to which the chronicles must not have significant similarities with each other.

By looking at the results one can argue that the better criteria for detecting change could be to look for the point in time where the difference between chronicle and sub-chronicle recognitions falls below the zero line. At that point, the number of recognition of sub-chronicle would be more as compared to that of original. This situation however demands significant occurrence of change. But the change can be small and difference, although still exists, could never fall below the zero line and we may not be able to detect the change. Consider the case shown in Figure 19 which shows the change detection process where there is small change. In this case, the change started to occur from 57th batch, but it never crossed the zero line because the change was not very significant.

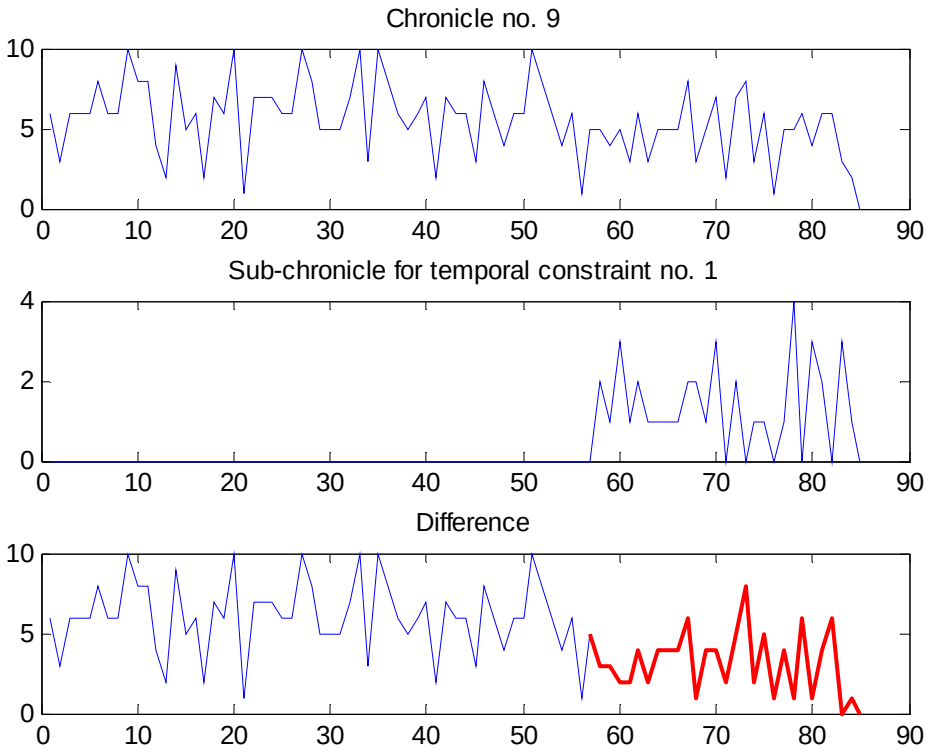


Figure 19 Graph showing change detection where difference never crossed the zero line

To conclude the discussion, we would like to emphasize that this was the first step in developing the approach for detecting concept change in the context of our framework. Although results are obtained using some assumption, but their correctness has encouraged us to continue the research in this direction.

6 Conclusion

In this research work, we introduced the notion of concept change and proposed three methods for detecting the concept change. We started by proposing the method based on finding percentage of completion but it offered a lot of drawbacks. This prompted us to introduce another method that used number of killings of largest partial instance at each temporal constraint for detecting concept change. We were interested in implementing this method but we faced some implementation issues while working with CRS software. Then the third technique was proposed that introduced the notion of sub-chronicles and we used their recognitions to detect the concept change. This technique can detect concept change if change occurs in only one of the temporal constraint of the chronicle.

After detecting the concept change, we proceeded toward overcoming it by performing chronicle adaptation. For that we introduced the time delay information which can guide the adaptation process. We are still in progress to develop a method for achieving adaptation using history of time delay information

There was no real life dataset available to us. So we implemented the change detection method on our own synthetic dataset and used CRS software for chronicle recognition. The presence of real life dataset however would have made the work more significant and would have helped us to discover different kind of real life problems that one can face in detecting concept change.

The results obtained using implemented techniques were quite good although there were based on some assumptions. But considering the fact that this was first step toward concept change detection in this framework, they have encouraged us working further on this technique so that it can be made more generalized.

References

[1] S. Cauvin, M.-O. Cordier, C. Dousson, P. Laborie, F. Lévy, J. Montmain, M. Porcheron, I. Servet, L. Travé-Massuyès, *Monitoring and alarm interpretation in industrial environments*, AI Communications 11 (3,4), 139-173, 1998

[2] C. Dousson, *Alarm driven supervision for telecommunication networks: II- On-line chronicle recognition*, Annals of Telecommunications n° 9/10 (tome 51), 501-508, 1996

[3] C. Dousson, P. Gaborit, M. Ghallab, *Situation recognition: representation and algorithms*, Proceedings of the 13th International Joint Conference on Artificial Intelligence, 166-172, 1993

[4] C. Dousson, P. Le Maigat, *Chronicle recognition improvement using temporal focusing and hierarchization*, Proceedings of the 20th International Conference on Artificial Intelligence, 324-329, 2007

[5] C. Dousson, *Extending and unifying chronicle representation with event counters,* Proceedings of the 15th International Conference on Artificial Intelligence, 257-261, 2002

[6] C. Dousson. *Crs: Chronicle recognition system.* http://crs.elibel.tm.fr/

[7] A. Tsymbal, *The problem of concept drift: definitions and related work*, Technical Report TCD-CS-2004-15, Department of Computer Science, Trinity College Dublin, Ireland, 2004

[8] Ying Yang, Xindong Wu, Xingquan Zhu, *Mining in Anticipation for Concept Change: Proactive-Reactive Prediction in Data Streams*, Data Mining and Knowledge Discovery, 13(3), 261-289, 2006

[9] F. Ferrer Troyano, J. Aguilar-Ruiz, and J. Riquelme, *Incremental rule learning and border examples selection from numerical data streams*, Journal of Universal Computer Science, 11(8), 1426-1439, 2005

[10] G. Widmer, M. Kubat, *Learning in the presence of concept drift and hidden contexts*, Machine Learning 23 (1), 69-101, 1996

[11] M.-O. Cordier, C. Dousson, *Alarm Driven Monitoring Based on Chronicles*, 4th Sumposium on Fault Detection Supervision and Safety for Technical Processes (SafeProcess), 286-291, 2000

[12] C. Dousson, Thang Vu Duong, *Discovering Chronicles with Numerical Time Constraints from Alarm Logs for Monitoring Dynamic Systems*, Proceedings of the 16th International Joint Conference on Artificial Intelligence, 620-626, 1999

[13] A. Osmani and F. Lévy, *A Constraint-Based Approach to Simulate Faults in Telecommunication Networks*, Proceedings of the 13th international conference on industrial and engineering applications of artificial intelligence and expert systems, 463-473, 2000

[14] S. Cauvin, B. Braunschweig, P. Galtier, Y. Glaize, *ALEXIP: an Expert System Coupled with a Dynamic Simulator for the Supervision of the Alphabutol Process*, Revue de l'Institut Français du Pétrole 47(3), 375-382, 1992

[15] C. Ungauer, *Problématique d'utilisation de techniques de supervision à base de connaissances profondes: L'exemple de la supervision du réseau TRANSPAC*, Rapport technique, CNET, Octobre 1993

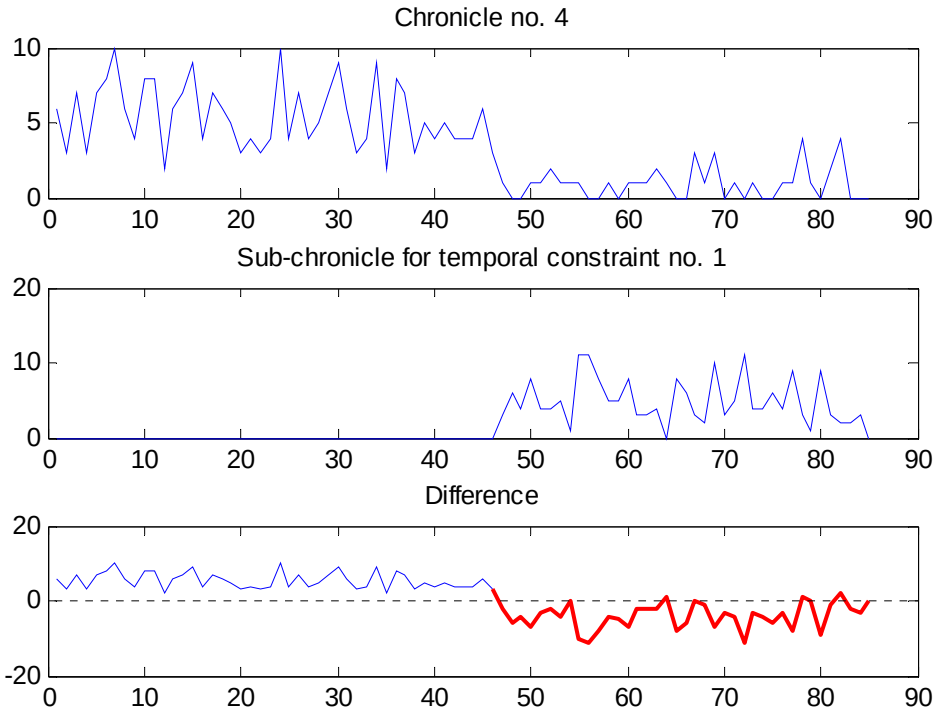
[16] S. Bibas, M.-O. Cordier, P. Dague, F. Lévy et L. Rozé, *Le projet GASPAR : Gestion d'alarmes par simulation des pannes sur le réseau Transpac*, Séminaire Gestion et Supervision de Réseaux, octobre 1995

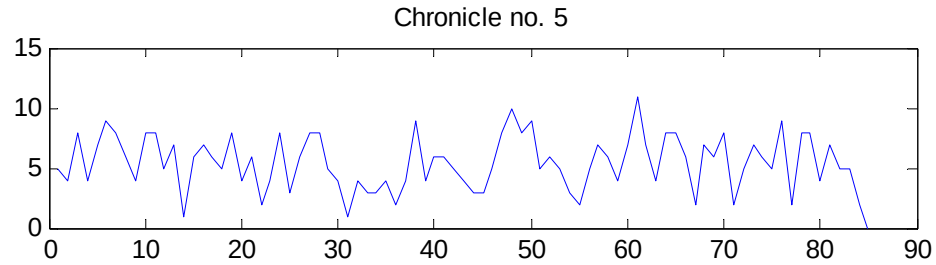
[17] P. Laborie and J.-P. Krivine, *Automatic generation of chronicles and its application to alarm processing in power distribution systems*, Proceedings of the International Workshop on Principles of Diagnosis, 61-68, 1997

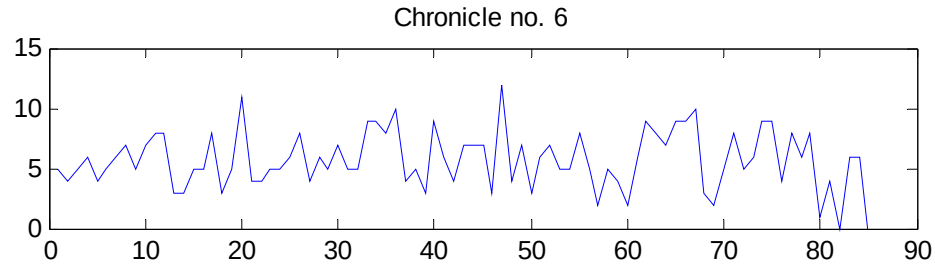
[18] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, *A discrete event systems approach to failure diagnosis,* International Workshop on Principles of Diagnosis, 269-277, 1994

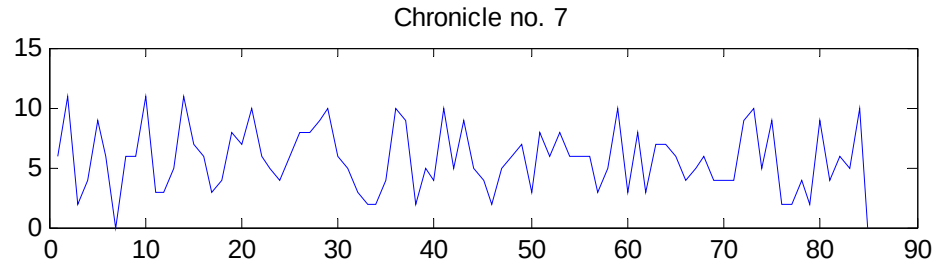
Appendix

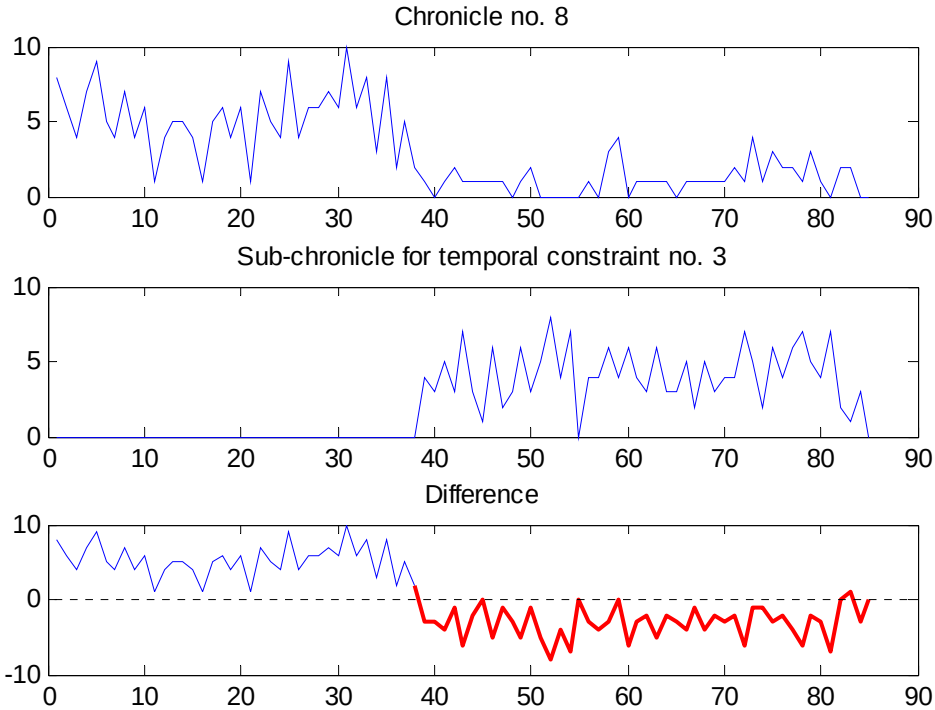
Simulation Results

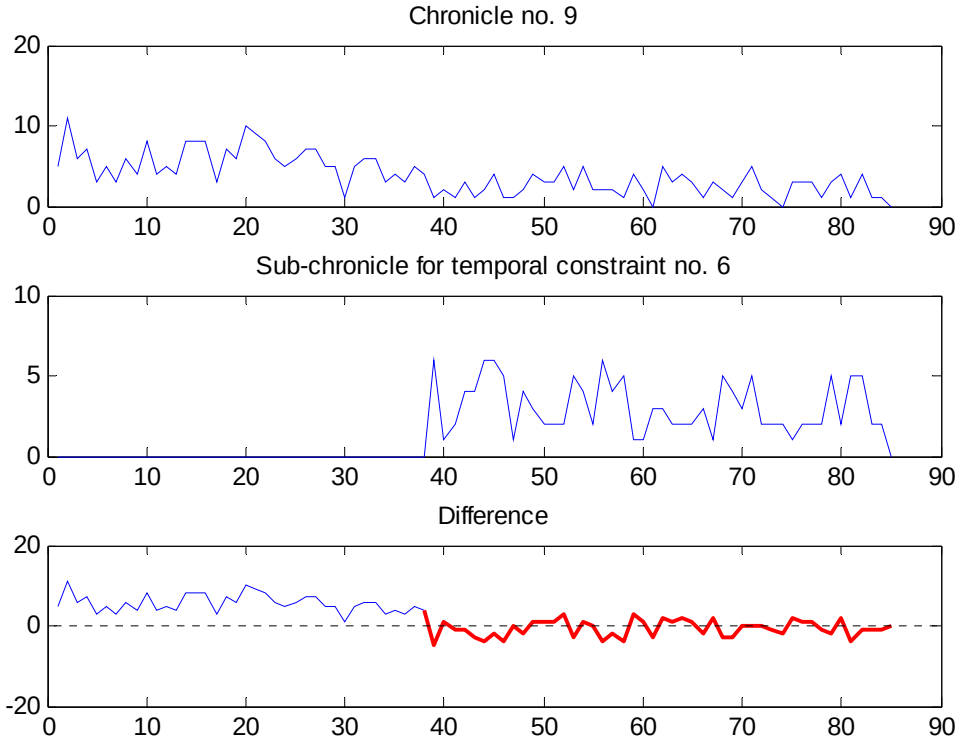


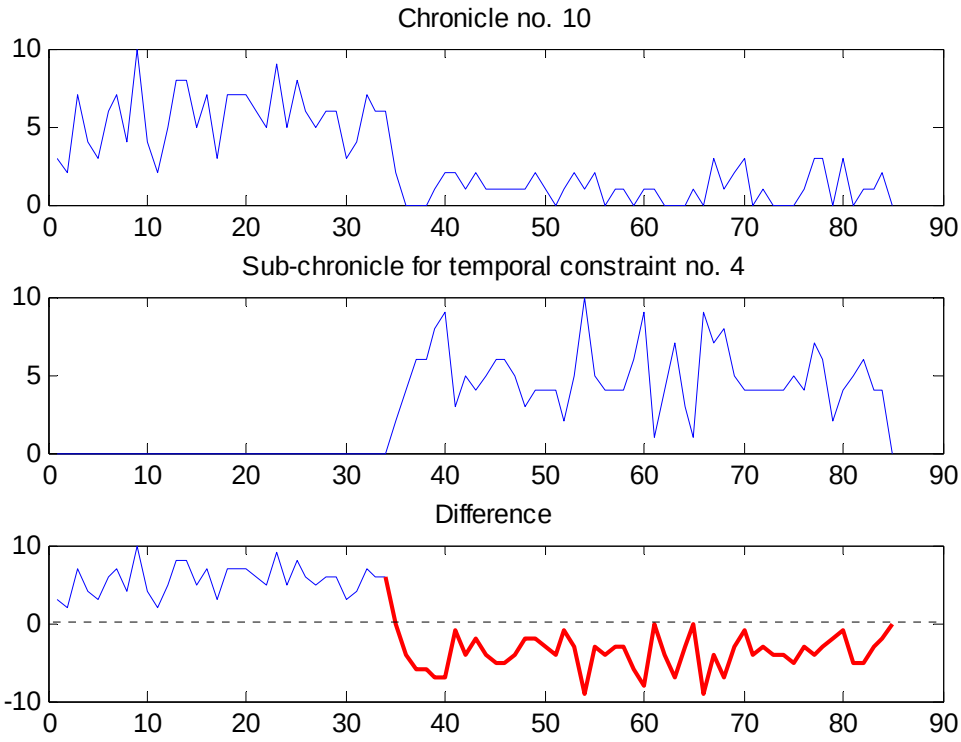


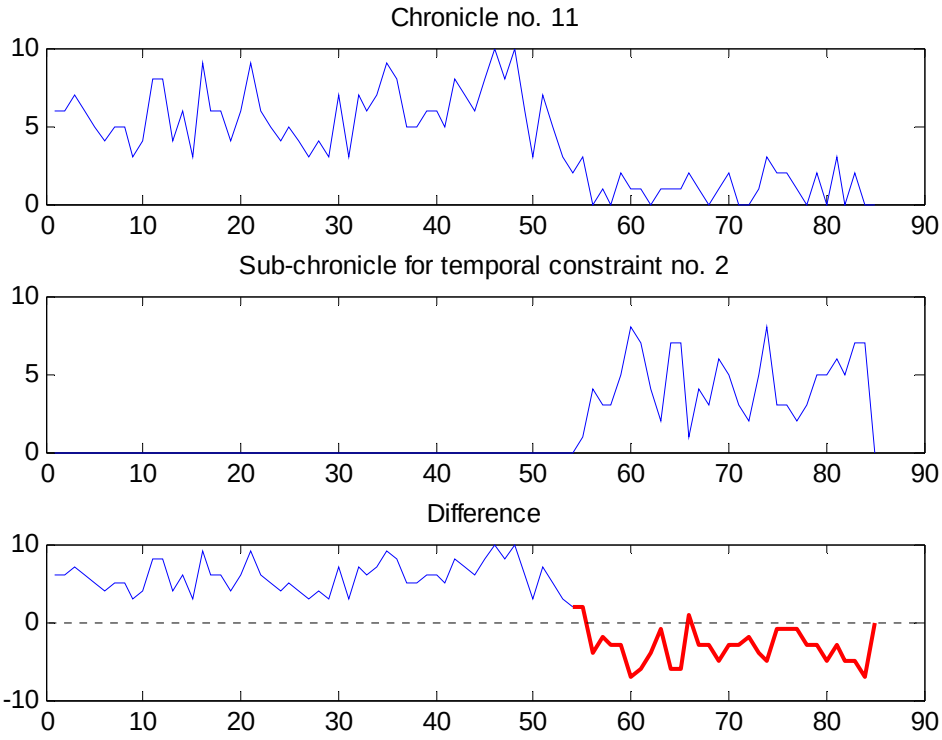

Chronicle no. 6
15
10
5
0
0
10
20
30
40
50
60
70
80
90

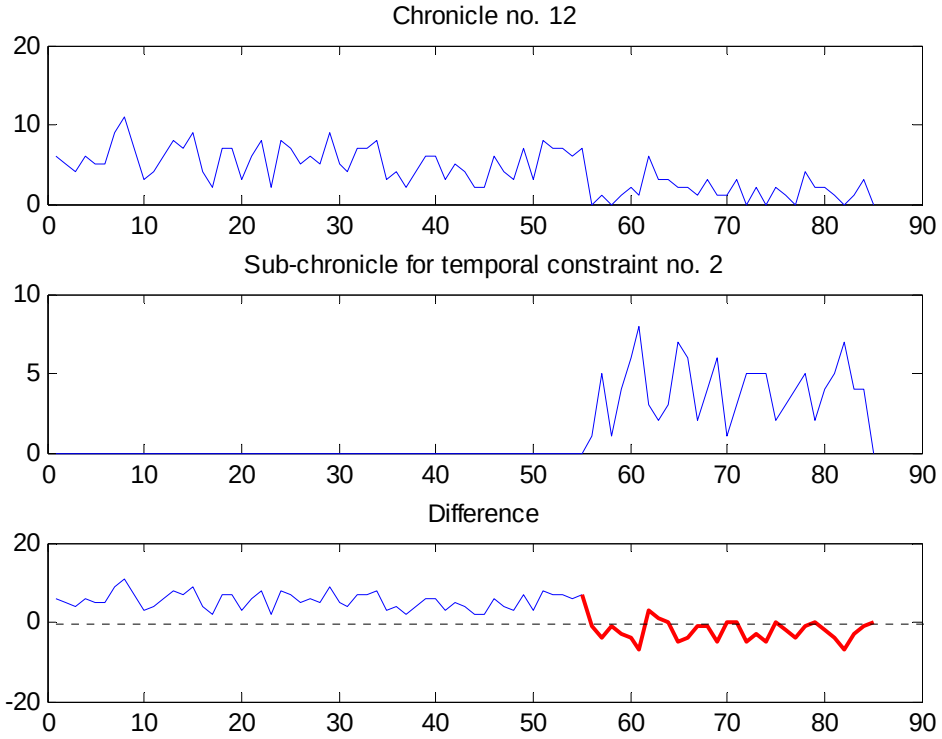

Chronicle no. 7
15
10
5
0
0
10
20
30
40
50
60
70
80
90


Chronicle no. 8
10
5
0
0
10
20
30
40
50
60
70
80
90
Sub-chronicle for temporal constraint no. 3
10
5
0
0
10
20
30
40
50
60
70
80
90
Difference
10
0
-10
0
10
20
30
40
50
60
70
80
90


Chronicle no. 9
Sub-chronicle for temporal constraint no. 6
Difference


Chronicle no. 10
Sub-chronicle for temporal constraint no. 4
Difference


Chronicle no. 11
Sub-chronicle for temporal constraint no. 2
Difference


Chronicle no. 12
Sub-chronicle for temporal constraint no. 2
Difference