

GMiner: User's guide (expanded)

From TDMiner

Contents

- 1 TDMiner
 - 1.1 Definitions
- 2 Features
- 3 Main Panel
- 4 Other Environments
- 5 Auxiliary Tools
- 6 Event Sequence Loader
- 7 Frequent Episode Miner
 - 7.1 Specify Episodes to Count
 - 7.2 Serial Episode Discovery
 - 7.3 Parallel Episode Discovery
 - 7.4 Specify Episodes to Count
 - 7.5 Serial Episode Discovery
 - 7.5.1 Fast Non-overlapped count(Serial):
 - 7.5.2 Non-overlapped count with episode expiry constraint(Serial):
 - 7.5.3 Non-overlapped count with inter-event expiry constraint(Serial):
 - 7.5.4 Non-overlapped count with inter-event interval constraint(Serial):
 - 7.5.5 Discovery of episodes & inter-event intervals(Serial):
 - 7.5.6 Discovery of generalized episodes(Serial):
 - 7.5.7 Discovery of generalized episodes with expiry constraint(Serial):
 - 7.5.8 Non-interleaved count(Serial):
 - 7.6 Parallel Episode Discovery
 - 7.6.1 Non-overlapped count(Parallel):
 - 7.6.2 Non-overlapped count with episode expiry constraint(Parallel):
- 8 Episode Visualization
- 9 Other Actions
- 10 Episode Simulation
- 11 Other Environments
- 12 Auxiliary Tools
 - 12.1 Tools
- 13 Event Data Stream
 - 13.1 Loading
 - 13.2 Prospecting
 - 13.3 Extraction
- 14 Usage
 - 14.1 Settings

TDMiner



TDMiner is a frequent episode mining tool useful to find interesting patterns or trends in large sequential data sets. The terminology used is introduced below before explaining how to use the tool.

Definitions

Data Mining

Data mining is the field concerned with analysis of large volumes of data to automatically unearth interesting information that is of value to the data owner. Interestingness may be defined as regularities or correlations in data or irregularities or unexpectedness depending on the nature of application. The regularities found in the data can be represented as association rules or clusters or recurrent patterns as in time series.

Temporal Data Mining

Temporal data mining is concerned with the mining of large sequential or ordered data streams. The data streams dealt with here are only categorical data sequences or event streams. The data can be viewed as a single long sequence of ordered pairs (E_i, t_i) which are called instantaneous events. In each event (E_i, t_i) , E_i is referred to as an event type (which takes its value from a finite alphabet) and t_i is the time of occurrence of the event. There are many applications where data appears in this form, e.g., alarm sequences in telecom networks, web navigation logs etc. However, there are also many situations where events tend to persist for different periods of time and in such cases the data is a sequence of events with both start as well as end times. The data here can be viewed as a sequence of ordered triplets (E_i, t_i, τ_i) , where E_i denotes the event type, t_i is the start time and τ_i is the end time of the event. (The data is ordered according to start times here).

Frequent Episodes

Frequent episodes are temporal patterns that occur sufficiently often (see *Frequency Threshold* below) along the data sequence. The temporal patterns referred to as episodes are ordered collections of event types. For example $A \rightarrow B \rightarrow C$ is a temporal pattern where an event type A is followed some time later by B and then a C , in that order. The frequent episode framework aims at discovering all episodes that occur often in the data.

Episode Structure

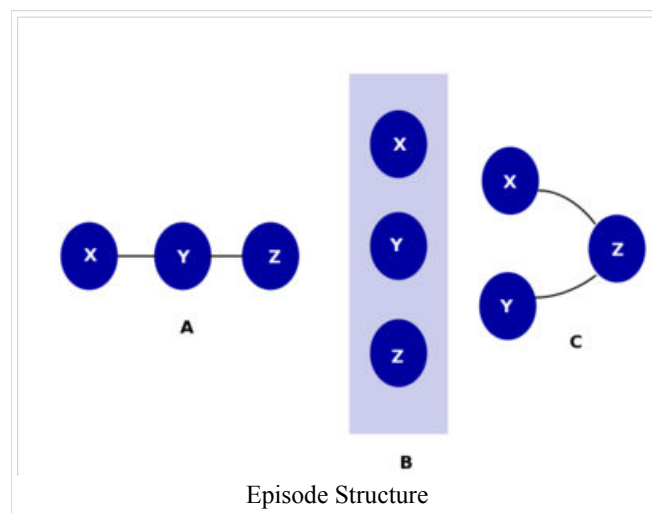
Three different kinds of episodes are shown in the figure below. Fig A shows a *serial episode*: it occurs in a data stream only if there are events of types X , Y and Z that occur in this order in the sequence. In the sequence there can be other events interspersed between the occurrences of event types of an episode. For Example, $X \rightarrow Y \rightarrow C \rightarrow Z$ is still an episode of X , Y and Z . Fig B shows a *parallel episode*: i.e. no constraints on the relative order of X , Y and Z are necessary. Fig C shows an example of a non-serial and non-parallel episode: it occurs in a sequence if events of type X and Y occur in any order followed some time later by an event of type Z .

Subepisode

An episode β is a subepisode of episode α if all event types of β are in α and if the partial order among the nodes of β is same as that for α .

Frequency Measure

The frequent episode framework aims at discovering all episodes that occur often in the data. In order to do this we need a frequency measure for episodes. Those episodes whose frequency count is greater than a user-specified threshold are declared as frequent episodes. The two frequency measures used in this tool are explained below.



Non-overlapped frequency: Two occurrences of an episode are said to be non-overlapped if no event associated with one appears in between the events associated with the other. The corresponding frequency for episode α is defined as the cardinality of the largest set of non-overlapped occurrences of α in the given event sequence. The definition of this frequency ensures that every subepisode is at least as frequent as the episode.

Example: Consider data: $\{(A, 1), (B, 5), (A, 6), (B, 7), (C, 8), (B, 10), (C, 11)\}$ and episode $A \rightarrow B \rightarrow C$. The occurrences $(A, 1), (B, 5), (C, 8)$ and $(A, 6), (B, 7), (C, 11)$ are not non-overlapped occurrences because $(A, 6)$ and $(B, 7)$ occur before the completion of the previous episode at $(C, 8)$. In this sequence there is not more than one non-overlapped occurrence. Hence the non-overlapped frequency count of the episode $A \rightarrow B \rightarrow C$ is one.

Non-interleaved frequency: Two occurrences of an episode are said to be non-interleaved if every event in the present occurrence of the episode, occurs not earlier than the occurrence of the next event in the previous occurrence of the episode. The corresponding frequency for episode α is defined as the cardinality of the largest set of non-interleaved occurrences of α in the given event sequence. In the above example there are two non-interleaved occurrences of the episode $A \rightarrow B \rightarrow C$: $(A, 1), (B, 5), (C, 8)$ and $(A, 6), (B, 10), (C, 11)$. Here $(A, 6)$ occurs after $(B, 5)$ and $(B, 10)$ occurs after $(C, 8)$. But $(A, 1), (B, 5), (C, 8)$ and $(A, 6), (B, 7), (C, 11)$ are not non-interleaved because $(B, 7)$ occurs before $(C, 8)$ has occurred in the previous occurrence. Non-interleaved frequency definition is only for serial episodes.

Frequency Threshold

An episode is said to be frequent if the frequency of that episode is greater than the user-specified *frequency threshold*. In this tool, the frequency threshold should be specified as a fraction of the data length. Thus, an episode is declared as frequent if its frequency exceeds the product of the frequency threshold and the total number of events in the data stream.

Episode Discovery

The frequent episode discovery framework usually employs a candidate generation technique because counting frequencies of all combinatorially possible episodes is too computationally expensive and hence infeasible in most applications. Frequent episode discovery is an iterative procedure where, in each iteration, through one pass over the data sequence, frequent episodes of a given size are found. These frequent episodes are used to find the candidate episodes of the next size and hence the frequent episodes of the next size. First, 1-node frequent episodes are found through one pass over the data. (This is same as a thresholding histogram of event types in the data). The frequent 1-node episodes are then combined (using a candidate generation algorithm) to obtain a set of 2-node

candidate episodes. By counting their frequencies through one pass over the data, frequent 2-node episodes are obtained. This process is continued till frequent episodes of all different sizes are obtained. The candidate generation strategies depend on the frequency measure used. Essentially, some necessary condition for an episode to be frequent is used so that the set of candidates is not too large. The frequency counting step depends on the frequency measure being used. In addition, we can also impose some constraints so that only those occurrences that satisfy these constraints are counted.

This tool can be used to find frequent episodes in large sequential data sets. It offers a choice of different frequency measures and their associated candidate generation strategies. It can be used to mine both serial and parallel episodes. Further, mining can be carried out with a variety of time constraints on the episodes. For example constraints can be placed on the time span of the episode, or time interval between two consecutive events in an episode etc. This tool can also be used for visualization of the data sequence and the episodes embedded in it. There are four simulator models to generate data sequences having embedded frequent episodes in them. Three of them are modeled to simulate multi-neural spike data and the fourth one is a generic data generator.

Features

The main user interface may be started in several ways. The most obvious is as a stand-alone application.

Main Panel

The main user interface of this tool has four main tabs: Use of this tool follows the functions identified under each of the main tabs tabs.

- Event Sequence Loader
- Frequent Episode Miner
- Episode Visualization
- Episode Simulation

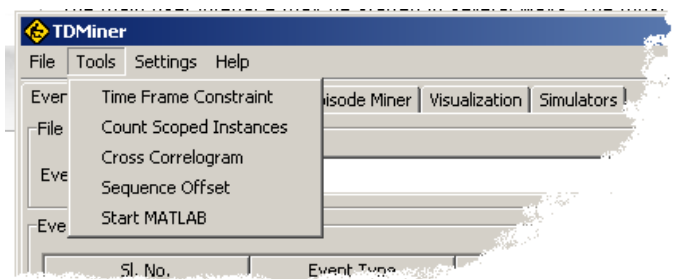
Other Environments

The various classes and objects may be accessed from any environment that has an interface to an appropriate Java virtual machine. An example of such an environment would be Matlab.

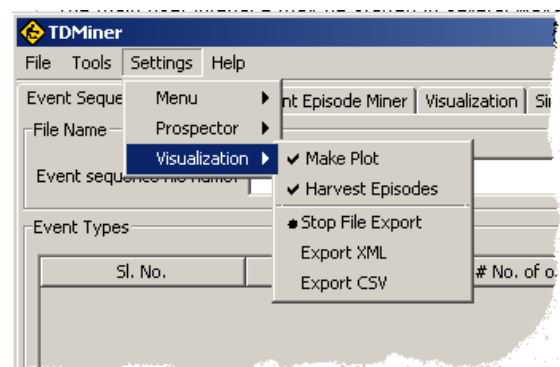
- GMiner: Matlab

Auxiliary Tools

- GMiner: Time frame constraints
- GMiner: Episode instance statistics



- GMiner: Exporting files
- GMiner: Save event stream
- GMiner: Save episode instances as events
- GMiner: Cross correlogram
- GMiner: Sequence offset



Event Sequence Loader


To load an Event Sequence:

The event file that is to be loaded should be in CSV (comma separated value) format. Each event is presented on a separate line. It is represented by the name of the event type followed by the time of occurrence separated by a comma. For the generalized event streams, each line has the event type, start time and end time separated by commas.

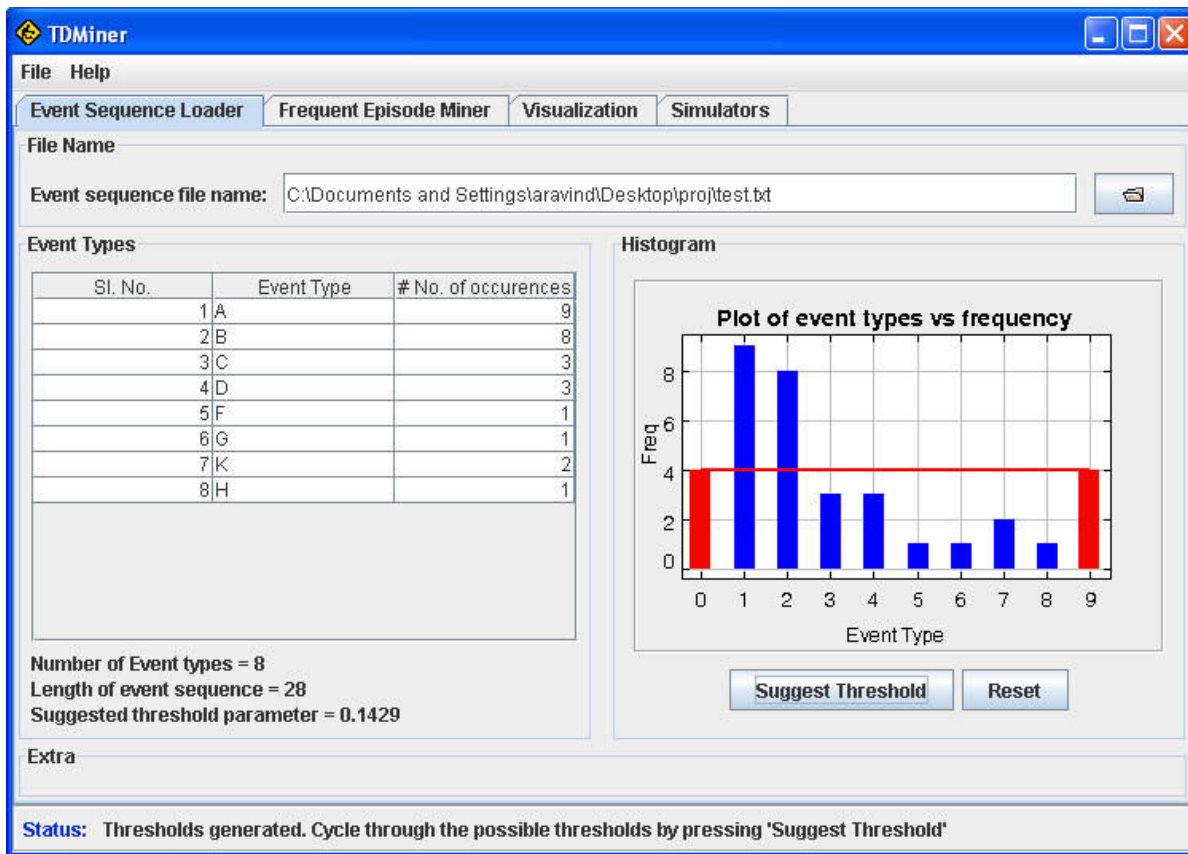
Examples:

Event Sequences	
Normal	Generalized
A,2	A, 2, 6
B,3	B, 4, 7
F,7	G, 7, 11

To load the event stream specify the path in the text box provided for '**Event sequence file name**'. (The Screen shot of the interface is given below).

Or click the  button to browse and select the file.

As soon as the file is loaded, the number of occurrences for each event type is presented automatically in the table at left hand side along with the histogram plot in the graph at right hand side of the window. Click the '**Suggest Threshold**' button to automatically obtain a frequency threshold. The frequency threshold is calculated from the histogram such that events that occur frequently are separated from events that occur sparsely. The '**Reset**' button restores the original magnification of the histogram plot.



Frequent Episode Miner

This allows for mining of different types of episodes with or without temporal constraints. Here, one has to choose the algorithm and various parameters needed by the algorithm.

The drop down menu giving the list of frequency counting algorithms is shown below.



The frequency counting algorithms available are

Specify Episodes to Count

- GMiner: Mining specified episodes

Serial Episode Discovery

- GMiner: Mining serial episodes using fast non-overlapped count
- GMiner: Mining serial episodes using non-overlapped count with episode expiry constraint
- GMiner: Mining serial episodes using non-overlapped count with inter-event expiry constraint
- GMiner: Mining serial episodes using non-overlapped count with inter-event interval constraint
- GMiner: Mining serial episodes using non-overlapped count with multiple inter-event intervals
- GMiner: Mining serial episodes using non-overlapped count for durable episodes
- GMiner: Mining durable serial episodes using non-overlapped count with episode expiry constraint
- GMiner: Mining serial episodes using non-interleaved count

Parallel Episode Discovery

- GMiner: Mining parallel episodes using non-overlapped count
- GMiner: Mining parallel episodes using non-overlapped count with episode expiry

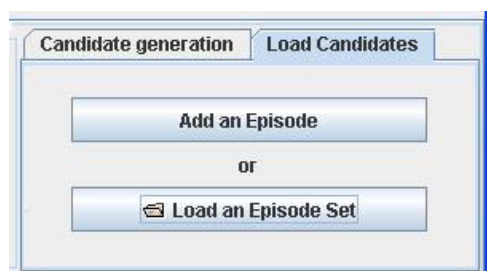
Specify Episodes to Count

To Specify an episode for counting:

The **Load Candidates** tab gives the option to obtain the count of a particular episode (or pattern) of interest.

1. Click the '**Add an Episode**' button to enter an episode and obtain its count. When adding an episode, separate each of its symbols with a space.
2. Click the '**Load an Episode Set**' button to load a previously saved episode set (using '**Save episodes**' button) for counting.

However this option of counting specific candidates under '**Load Candidates**' tab is not available for, **Discovery of episodes & inter-event intervals(Serial)**, **Discovery of generalized episodes(Serial)**, **Discovery of generalized episodes with expiry constraint(Serial)** algorithms.



Serial Episode Discovery

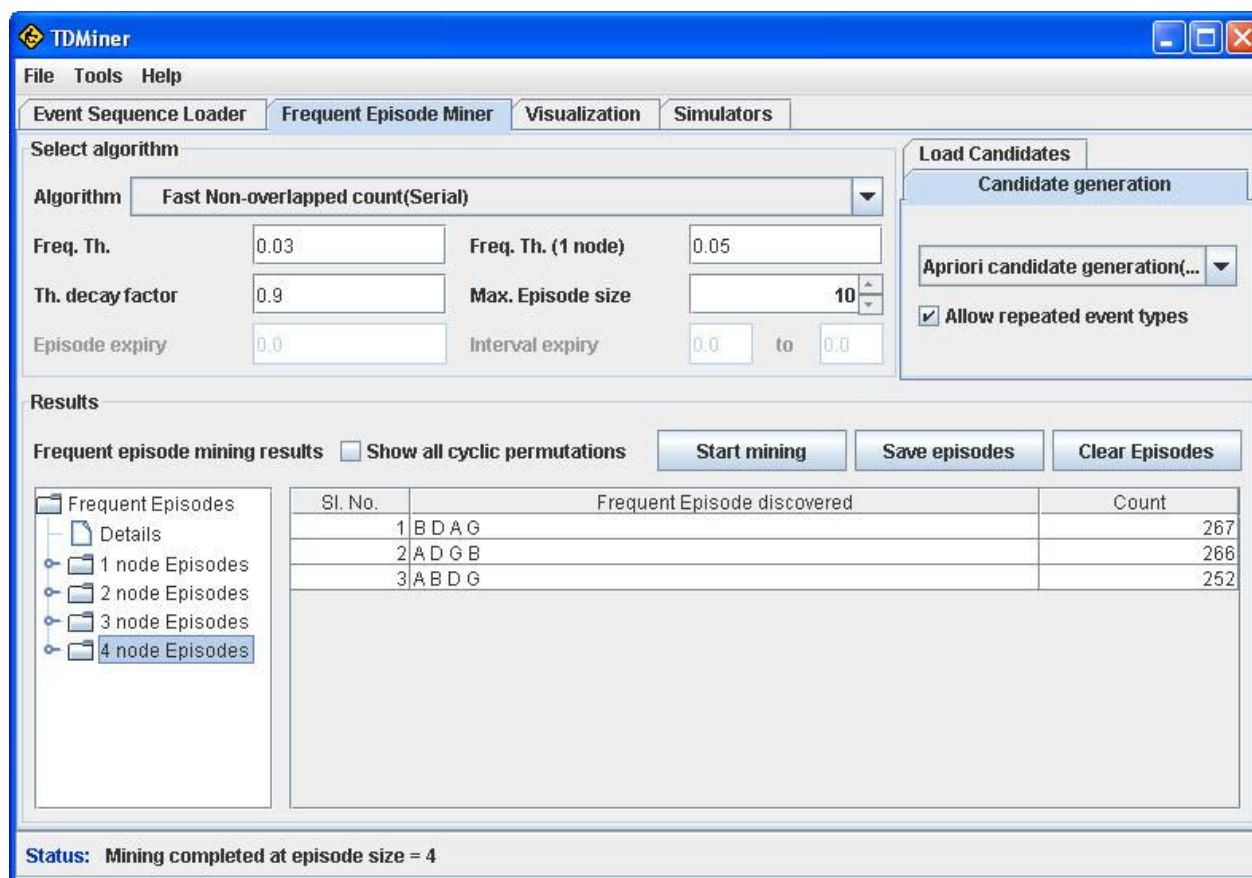
Fast Non-overlapped count(Serial):

This allows for fast mining of serial episodes. However, this algorithm will not allow any temporal constraints to be imposed. The following steps are to be followed to mine episodes using this algorithm.

1. Load the data sequence using '**Event Sequence Loader**' or generate a data sequence using an appropriate model in '**Simulators**'.
2. Choose the '**Fast Non-overlapped count(Serial)**' algorithm and the corresponding candidate generation algorithm '**Apriori candidate generation(Serial)**' gets automatically selected.

3. Uncheck the '**Allow repeated event types**' option to avoid mining of episodes that have repeated events.
4. Enter the values of '**Freq. Th. (1 node)**' which is the frequency threshold used for one node episodes and '**Freq. Th.**' which is the frequency threshold used for episodes with more than one node. (These values can be automatically set by clicking the '**Suggest Threshold**' button in '**Event Sequence Loader**').
5. Enter the value of '**Th. decay factor**'. The frequency threshold gets decayed by this factor after every iteration from two node episode onwards. (The default value is 1)
6. Enter the value of '**Max Episode size**'. It is the maximum size of the episode that will be mined. (The default value is 10)
7. Click the '**Start mining**' button to get the list of frequent episodes of different sizes.

Click the '**Save episodes**' button to save the list of frequent episodes into a file. Note that, at present, information about the analysis that produced this result is not recorded. Click the '**Clear Episodes**' button to clear the list of frequent episodes. Uncheck the '**Show all cyclic permutations**' option to avoid listing those episodes which are cyclic permutation of another frequent episode.



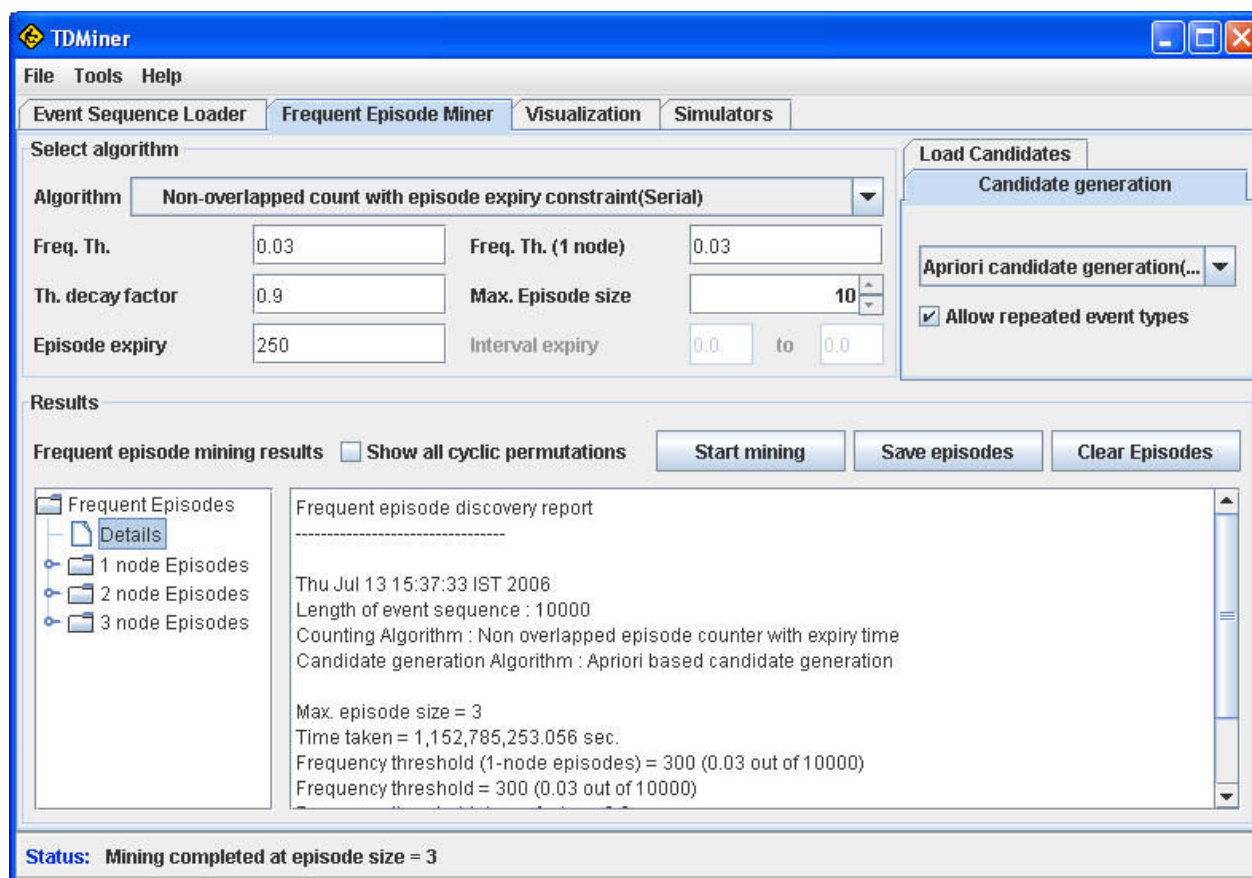
Non-overlapped count with episode expiry constraint(Serial):

This algorithm mines only those episodes that satisfy the episode expiry constraint specified. In any occurrence of an episode, the difference between the times of the last and first events constituting this occurrence is called the span of the occurrence. Under the episode expiry constraint, only those occurrences whose span is less than or equal to the expiry time specified by the user would be counted. The following steps are to be followed to mine episodes using this algorithm.

1. Load the data sequence using '**Event Sequence Loader**' or generate a data sequence using an appropriate model in '**Simulators**'.
2. Choose the '**Non-overlapped count with episode expiry constraint(Serial)**' algorithm and the corresponding candidate generation algorithm '**Apriori candidate generation(Serial)**' gets automatically selected.
3. Uncheck the '**Allow repeated event types**' option to avoid mining of episodes which have repeated events.

4. Enter the values of '**Freq. Th. (1 node)**' which is the frequency threshold used for one node episodes and '**Freq. Th.**' which is the frequency threshold used for episodes with more than one node. (These values can be automatically set using '**Suggest Threshold**' button in '**Event Sequence Loader**').
5. Enter the value of '**Th. decay factor**'. The frequency threshold gets decayed by this factor after every iteration from two node episode onwards. (The default value is 1).
6. Enter the value of '**Max. Episode size**'. It is the maximum size of the episode that will be mined. (The default value is 10).
7. Enter the value of '**Episode expiry**'. This specifies the maximum span of an episode.
8. Click the '**Start mining**' button to get the list of frequent episodes of different sizes.

Click the '**Save episodes**' button to save the list of frequent episodes into a file. Click the '**Clear Episodes**' button to clear the list of frequent episodes. Uncheck the '**Show all cyclic permutations**' option to avoid listing those episodes which are cyclic permutation of another frequent episode.



Non-overlapped count with inter-event expiry constraint(Serial):

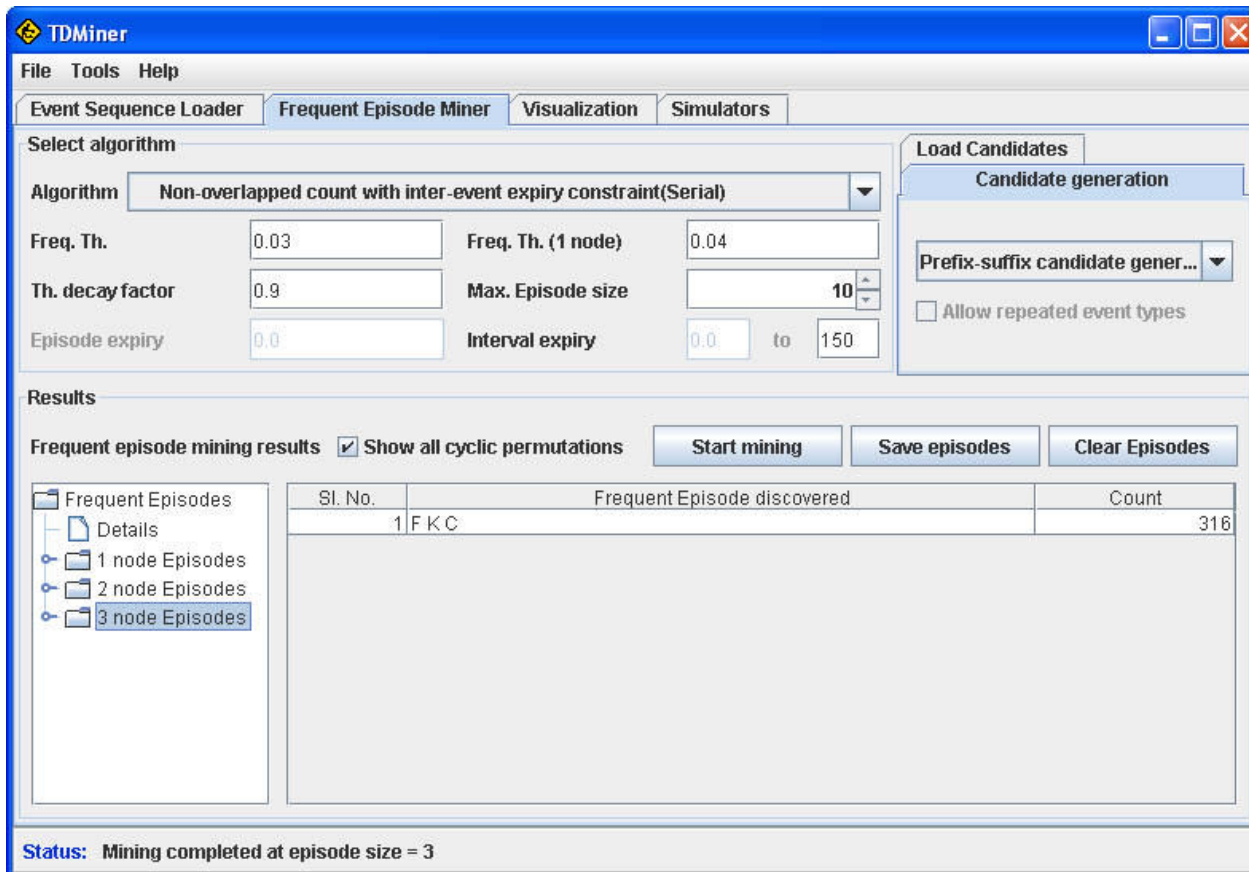
This algorithm mines episodes with inter-event expiry constraint. The constraint here specifies the maximum allowable time between the successive events of the episode. Suppose (A, t1), (B, t2) and (C, t3) constitute an occurrence of the serial episode A→B→C. Then (t2-t1) and (t3-t2) are the inter-event times for this occurrence. Under inter-event expiry constraint, an occurrence is counted only if the inter-event times are less than or equal to the Interval Expiry time specified by the user. The following steps are to be followed to mine episodes using this algorithm.

1. Load the data sequence using '**Event Sequence Loader**' or generate a data sequence using an appropriate model in '**Simulators**'.
2. Choose the '**Non-overlapped count with inter-event expiry constraint(Serial)**' algorithm and the corresponding candidate generation algorithm '**Prefix-suffix candidate generation**' gets automatically selected.
3. Enter the values of '**Freq. Th. (1 node)**' which is the frequency threshold used for one node episodes and '**Freq. Th.**' which is the frequency threshold used for episodes with more than one node. (These values can be

automatically set using '**Suggest Threshold**' button in '**Event Sequence Loader**'.

4. Enter the value of '**Th. decay factor**'. The frequency threshold gets decayed by this factor after every iteration from two node episode onwards. (The default value is 1.)
5. Enter the value of '**Max. Episode size**'. It is the maximum size of the episode that will be mined. (The default value is 10)
6. Enter the value of '**Interval Expiry**'. The inter-event times in all occurrences of the episode that are counted would be less than or equal to this value.
7. Click the '**Start mining**' button to get the list of frequent episodes of different sizes.

Click the '**Save episodes**' button to save the list of frequent episodes into a file. Click the '**Clear Episodes**' button to clear the list of frequent episodes. Uncheck the '**Show all cyclic permutations**' option to avoid listing those episodes which are cyclic permutation of another frequent episode.



Non-overlapped count with inter-event interval constraint(Serial):

This algorithm mines serial episodes with an interval constraint for inter-event times. The constraint specifies an interval for inter-event times. In all occurrences counted by the algorithm, the inter-event times would be in this interval. Unlike the inter-event expiry constraint where we have only an upper bound on the inter-event times, here we have both an upper bound and a lower bound for inter-event times. The following steps are to be followed to mine episodes using this algorithm.

1. Load the data sequence using '**Event Sequence Loader**' or generate a data sequence using an appropriate model in '**Simulators**'.
2. Choose the '**Non-overlapped count with inter-event interval constraint(Serial)**' algorithm and the corresponding candidate generation algorithm '**Prefix-suffix candidate generation**' gets automatically selected.
3. Enter the values of '**Freq. Th. (1 node)**' which is the frequency threshold used for one node episodes and '**Freq. Th.**' which is the frequency threshold used for episodes with more than one node. (These values can be automatically set using '**Suggest Threshold**' button in '**Event Sequence Loader**'.

4. Enter the value of '**Th. decay factor**'. The frequency threshold gets decayed by this factor after every iteration from two node episode onwards. (The default value is 1)
5. Enter the value of '**Max. Episode size**'. It is the maximum size of the episode that will be mined. (The default value is 10)
6. Enter the lower and higher value of '**Interval Expiry**'. In every occurrence of the episode counted by the algorithm, the inter-event times would be in this interval.
7. Click the '**Start mining**' button to get the list of frequent episodes of different sizes.

Click the '**Save episodes**' button to save the list of frequent episodes into a file. Click the '**Clear Episodes**' button to clear the list of frequent episodes. Uncheck the '**Show all cyclic permutations**' option to avoid listing those episodes which are cyclic permutation of another frequent episode.

Sl. No.	Frequent Episode discovered	Count
1	G E D	80
2	D E B	77
3	D H E	75
4	H E G	75
5	H E D	71

Status: Mining completed at episode size = 3

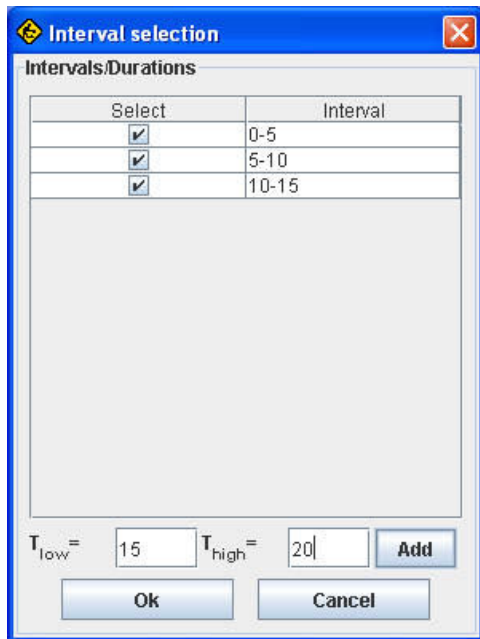
Discovery of episodes & inter-event intervals(Serial):

This algorithm not only discovers frequent episodes but also discovers the inter-event intervals with which consecutive events in the episode occur frequently. Suppose $A \rightarrow C \rightarrow D$ is a frequent episode. This means that, often, A is followed sometime later by C which is followed by D. The inter-event times in these occurrences might also be following a pattern depending upon the nature of the process generating the data. If the timing information between consecutive events is also of interest, then this algorithm is to be used. A list of intervals has to be entered which denote all the possible inter-event interval constraints. (These may be guessed with some knowledge of the data or the process generating the data). The output will be a list of frequent episodes along with an interval for every inter-event time. Each such interval would be one of the intervals specified by the user. Suppose the set of intervals specified are (0-5), (5-10), (10-20). Then, one of the output frequent episodes could be $A(0-5) \rightarrow B(10-20) \rightarrow C$. This means that there are a sufficiently large number (depending on the frequency threshold) of occurrences of $A \rightarrow B \rightarrow C$ with the inter-event times between A and B being in the interval (0,5) and the inter-event times between B and C being in the interval (10,20). The algorithm searches for the interval possibilities only among the intervals specified by the user. The following steps are to be followed to mine episodes using this algorithm.

1. Load the data sequence using '**Event Sequence Loader**' or generate a data sequence using an appropriate model in

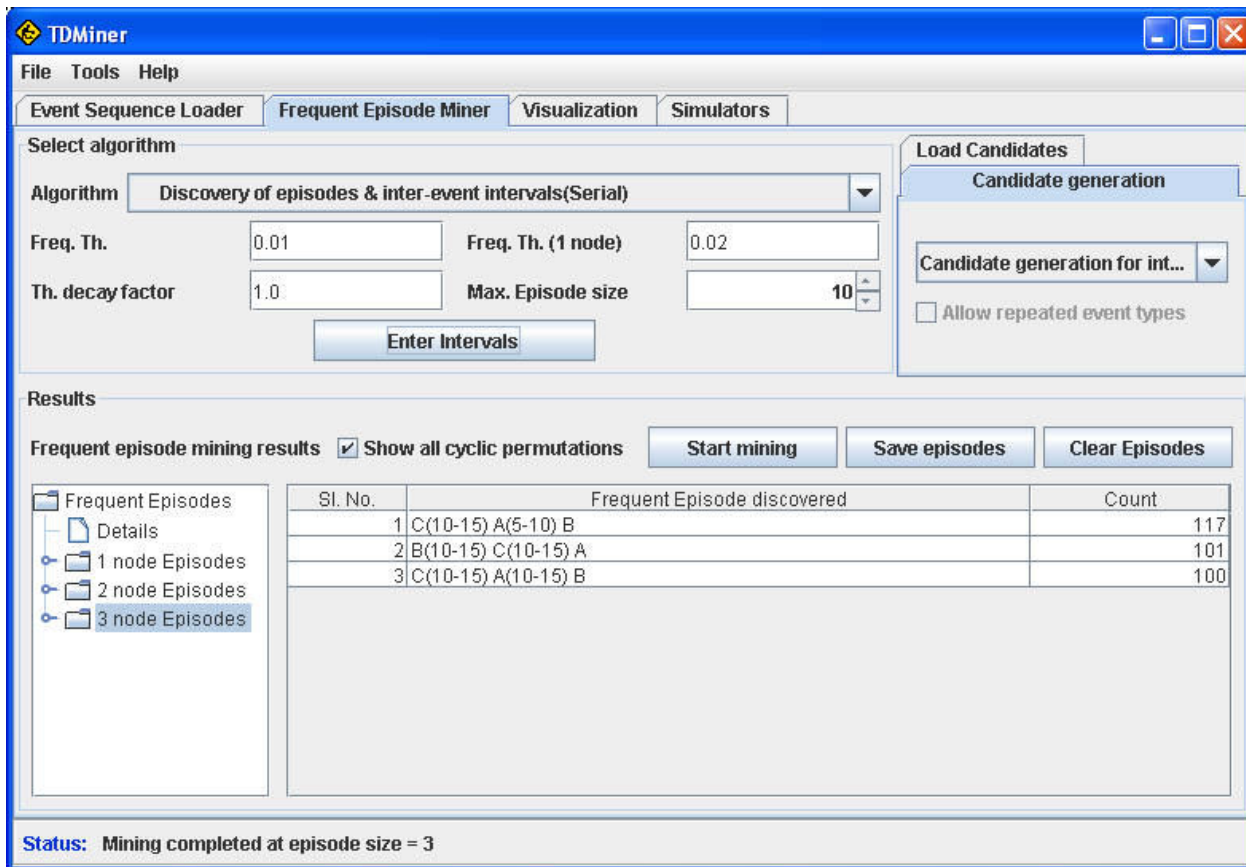
'Simulators'.

2. Choose the **'Discovery of episodes & inter-event intervals(Serial)'** algorithm and the corresponding candidate generation algorithm **'Candidate generation for interval discovery'** gets automatically selected.
3. Enter the values of **'Freq. Th. (1 node)'** which is the frequency threshold used for one node episodes and **'Freq. Th.'** which is the frequency threshold used for episodes with more than one node. (These values can be automatically set using **'Suggest Threshold'** button in **'Event Sequence Loader'**).
4. Enter the value of **'Th. decay factor'**. The frequency threshold gets decayed by this factor after every iteration from two node episode onwards. (The default value is 1)
5. Enter the value of **'Max. Episode size'**. It is the maximum size of the episode that will be mined. (The default value is 10)
6. Click the **'Enter Intervals'** button to enter the intervals in a new window that opens as shown below. Enter the **'T_{low}'** and **'T_{high}'** value of the interval and click **'Add'** button to enter the interval. To remove an already added interval uncheck the checkbox corresponding to that interval. Finally click the **'Ok'** button after all the intervals have been entered.



7. Click the **'Start mining'** button to get the list of frequent episodes of different sizes along with the discovered inter-event times.

Click the **'Save episodes'** button to save the list of frequent episodes into a file. Click the **'Clear Episodes'** button to clear the list of frequent episodes.



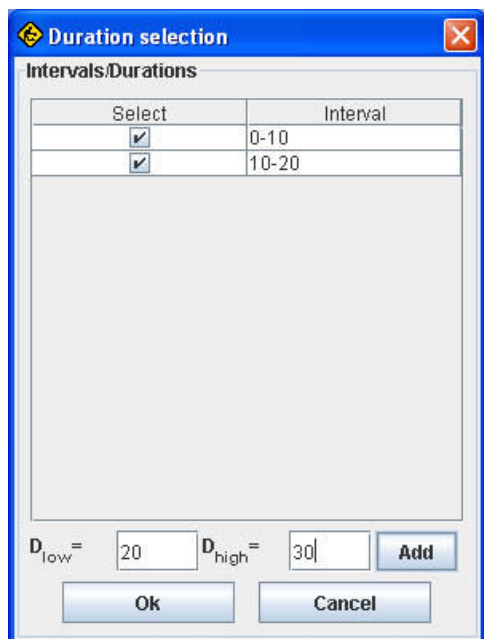
Discovery of generalized episodes(Serial):

The events in the data stream here have both start times and end times, i.e. events persist for different periods of time and are not instantaneous. Generalized episodes not only specify an order on event types but also some constraints on the durations of events. A generalized episode is represented as

$A[5\ 10] \rightarrow B[0\ 5, 10\ 15] \rightarrow C[0\ 5]$

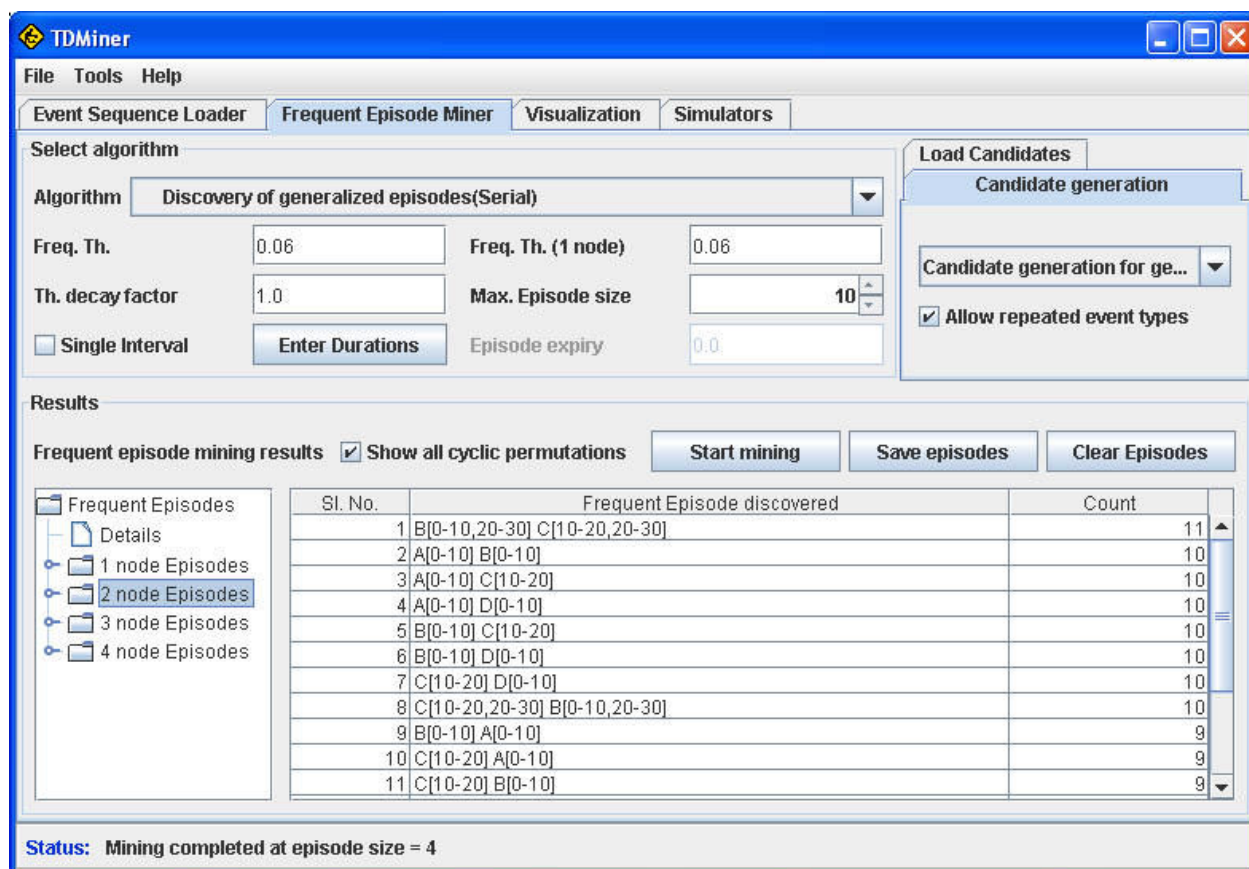
For an occurrence of this generalized episode we need an event of type A whose duration is in the time interval [5, 10] followed by an event of type B whose duration is either in the interval [0, 5] or in the interval [10, 15] which is then followed by an event of type C with duration in the interval [0, 5]. To discover frequent generalized episodes, the user has to provide a list of non overlapping time intervals (like [0 5], [5 10]). The algorithm discovers frequent generalized episodes where the time duration possibilities tagged with each node would be a union of one or more intervals specified by the user. The following steps are to be followed to mine episodes using this algorithm.

1. Load the data sequence using '**Event Sequence Loader**' .
2. Choose the '**Discovery of generalized episodes(Serial)**' algorithm and the corresponding candidate generation algorithm '**Candidate generation for generalized episodes**' gets automatically selected.
3. Enter the values of '**Freq. Th. (1 node)**' which is the frequency threshold used for one node episodes and '**Freq. Th.**' which is the frequency threshold used for episodes with more than one node. (These values can be automatically set using '**Suggest Threshold**' button in '**Event Sequence Loader**'.
4. Enter the value of '**Th. decay factor**'. The frequency threshold gets decayed by this factor after every iteration from two node episode onwards. (The default value is 1)
5. Enter the value of '**Max. Episode size**'. It is the maximum size of the episode that will be mined. (The default value is 10)
6. Click the '**Enter Durations**' button to enter the durations in a new window that opens as shown below. Enter the '**D_{low}**' and '**D_{high}**' value of the durations and click '**Add**' button to enter the duration. To remove an already added duration uncheck the checkbox corresponding to that duration. Finally click '**Ok**' button after all the intervals have been entered.



- Click the '**Start mining**' button to get the list of frequent episodes of different sizes along with the discovered inter-event times.

Click the '**Save episodes**' button to save the list of frequent episodes into a file. Click the '**Clear Episodes**' button to clear the list of frequent episodes.



Discovery of generalized episodes with expiry constraint(Serial):

This algorithm discovers generalized frequent episodes under episode expiry constraint. That is, all occurrences of the episode counted by the algorithms are such that the span of the occurrence is less than the expiry time specified. The following steps are to be followed to mine episodes using this algorithm.

1. Load the data sequence using '**Event Sequence Loader**' .
2. Choose the '**Discovery of generalized episodes with expiry constraint(Serial)**' algorithm and the corresponding candidate generation algorithm '**Candidate generation for generalized episodes**' gets automatically selected.
3. Enter the values of '**Freq. Th. (1 node)**' which is the frequency threshold used for one node episodes and '**Freq. Th.**' which is the frequency threshold used for episodes with more than one node. (These values can be automatically set using '**Suggest Threshold**' button in '**Event Sequence Loader**').
4. Enter the value of '**Th. decay factor**'. The frequency threshold gets decayed by this factor after every iteration from two node episode onwards. (The default value is 1)
5. Enter the value of '**Max. Episode size**'. It is the maximum size of the episode that will be mined. (The default value is 10)
6. Click the '**Enter Durations**' button to enter the durations in a new window that opens as shown below. Enter the '**D_{low}**' and '**D_{high}**' value of the durations and click '**Add**' button to enter the duration. To remove an already added duration uncheck the checkbox corresponding to that duration. Finally click '**Ok**' button after all the intervals have been entered.

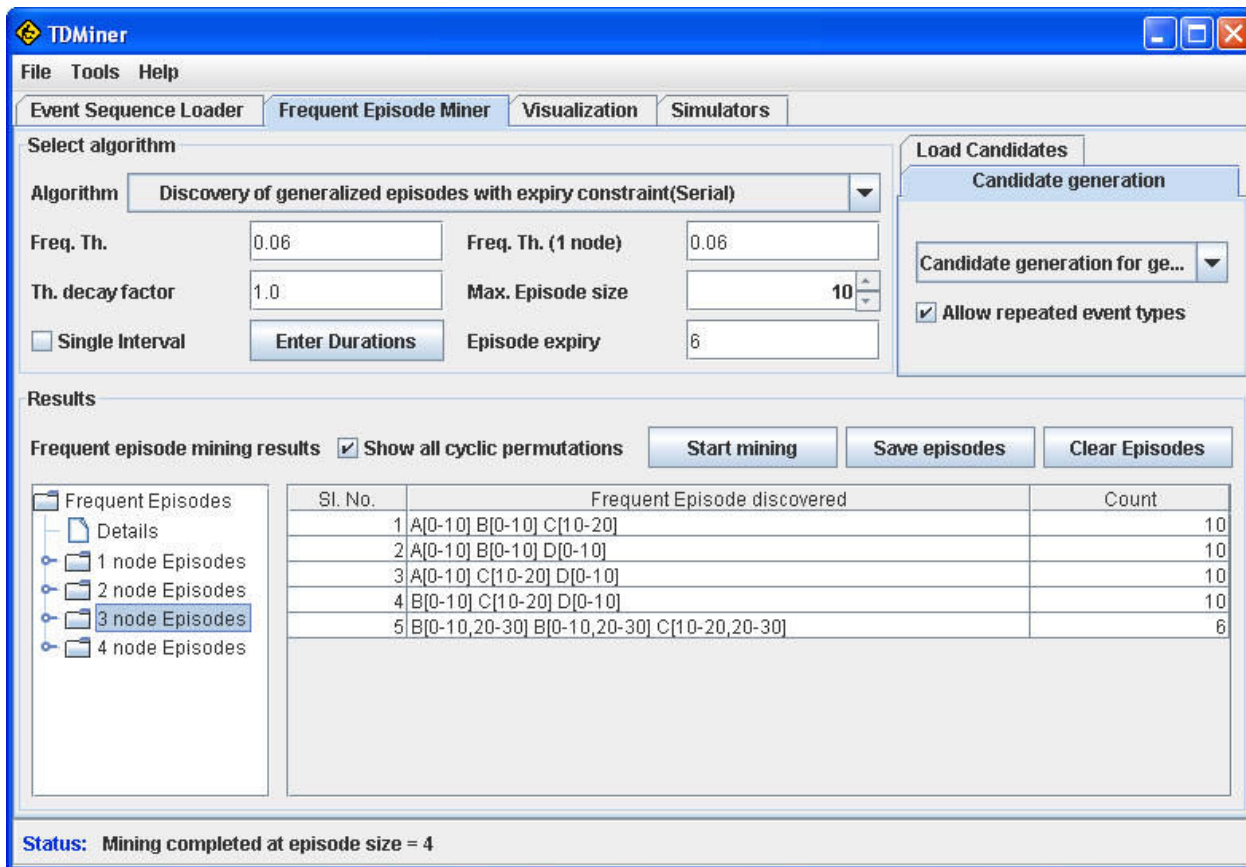
Select	Interval
<input checked="" type="checkbox"/>	0-10
<input checked="" type="checkbox"/>	10-20

D_{low} = 20 D_{high} = 30 [Add]

[Ok] [Cancel]

7. Enter the value of '**Episode expiry**'.
8. Click the '**Start mining**' button to get the list of frequent episodes of different sizes along with the discovered inter-event times.

Click the '**Save episodes**' button to save the list of frequent episodes into a file. Click the '**Clear Episodes**' button to clear the list of frequent episodes.



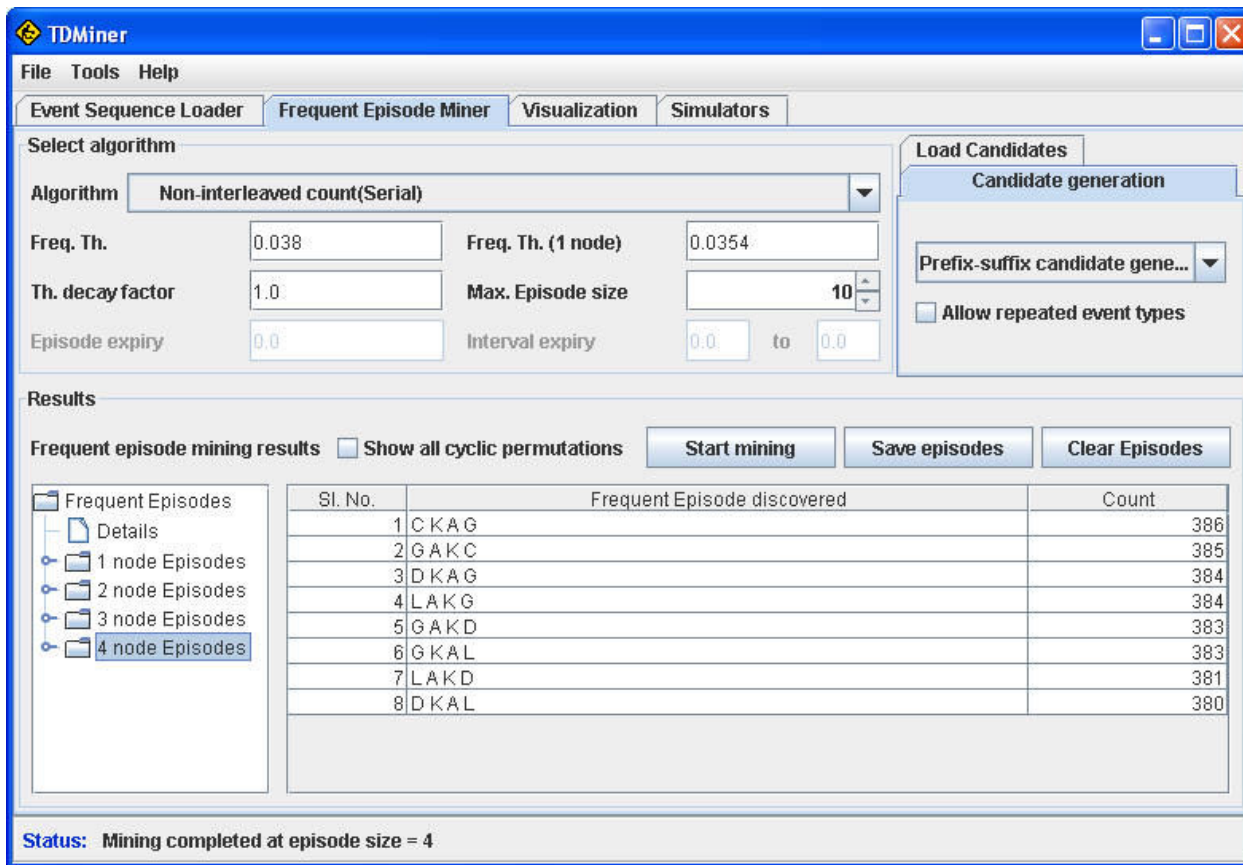
Non-interleaved count(Serial):

This algorithm is for discovering frequent serial episodes where the frequency measure used is the number of non-interleaved occurrences. This algorithm cannot incorporate any temporal constraints on the episodes. The following steps are to be followed to mine non-interleaved episodes using this algorithm.

1. Load the data sequence using '**Event Sequence Loader**' or generate a data sequence using an appropriate model in '**Simulators**'.
2. Choose the '**Non-interleaved count(Serial)**' algorithm and the corresponding candidate generation algorithm '**Prefix-suffix candidate generation**' gets automatically selected.
3. Uncheck the '**Allow repeated event types**' option to avoid mining of episodes which have repeated events.
4. Enter the values of '**Freq. Th. (1 node)**' which is the frequency threshold used for one node episodes and '**Freq. Th.**' which is the frequency threshold used for episodes with more than one node. (These values can be automatically set by clicking the '**Suggest Threshold**' button in '**Event Sequence Loader**'.
5. Enter the value of '**Th. decay factor**'. The frequency threshold gets decayed by this factor after every iteration from two node episode onwards. (The default value is 1)
6. Enter the value of '**Max. Episode size**'. It is the maximum size of the episode that will be mined. (The default value is 10)
7. Click the '**Start mining**' button to get the list of frequent episodes of different sizes.

Click the '**Save episodes**' button to save the list of frequent episodes into a file. Click the '**Clear Episodes**' button to clear the list of frequent episodes. Uncheck the '**Show all cyclic permutations**' option to avoid listing those episodes which are cyclic permutation of another

frequent episode.



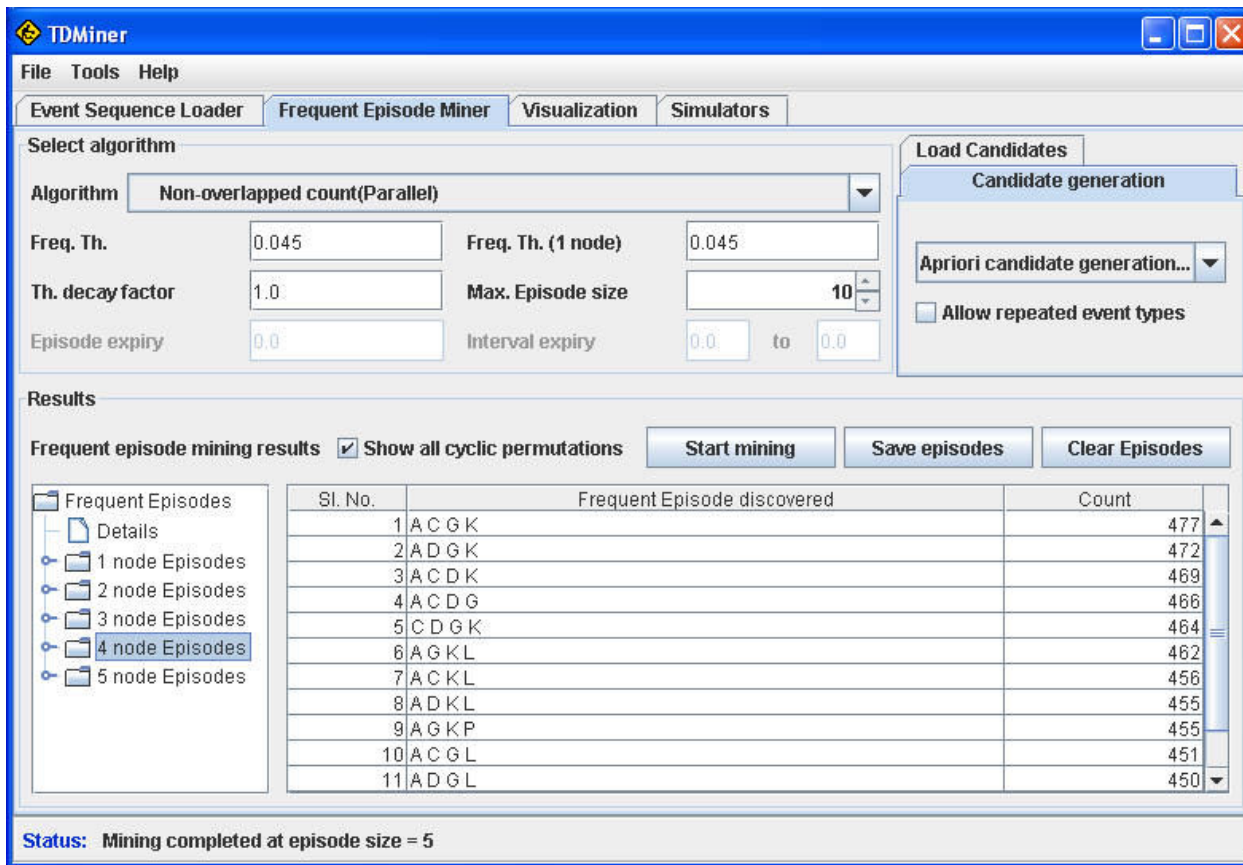
Parallel Episode Discovery

Non-overlapped count(Parallel:

All the algorithms considered so far are for serial episodes. This algorithm discovers frequent parallel episodes. The frequency measure is the number of non-overlapped occurrences. Recall that in a parallel episode the events constituting an occurrence can be in any temporal order. The following steps are to be followed to mine parallel episodes using this algorithm.

1. Load the data sequence using '**Event Sequence Loader**' or generate a data sequence using an appropriate model in '**Simulators**'.
2. Choose the '**Non-overlapped count(Parallel)**' algorithm and the corresponding candidate generation algorithm '**Apriori candidate generation(Parallel)**' gets automatically selected.
3. Uncheck the '**Allow repeated event types**' option to avoid mining of episodes which have repeated events.
4. Enter the values of '**Freq. Th. (1 node)**' which is the frequency threshold used for one node episodes and '**Freq. Th.**' which is the frequency threshold used for episodes with more than one node. (These values can be automatically set by clicking the '**Suggest Threshold**' button in '**Event Sequence Loader**'.
5. Enter the value of '**Th. decay factor**'. The frequency threshold gets decayed by this factor after every iteration from two node episode onwards. (The default value is 1)
6. Enter the value of '**Max. Episode size**'. It is the maximum size of the episode that will be mined. (The default value is 10)
7. Click the '**Start mining**' button to get the list of frequent episodes of different sizes.

Click the '**Save episodes**' button to save the list of frequent episodes into a file. Click the '**Clear Episodes**' button to clear the list of frequent episodes. Uncheck the '**Show all cyclic permutations**' option to avoid listing those episodes which are cyclic permutation of another frequent episode.

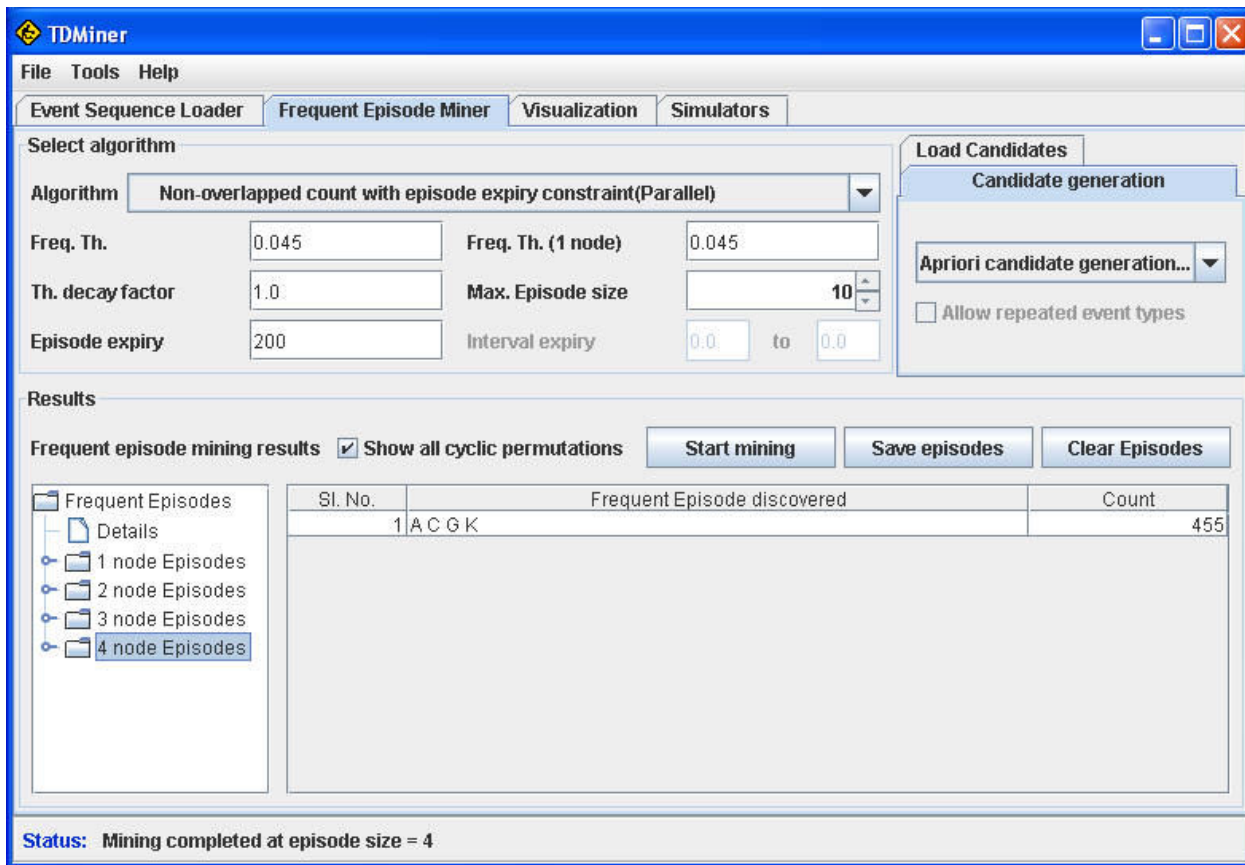


Non-overlapped count with episode expiry constraint(Parallel):

This algorithm mines parallel episodes with an episode expiry constraint specified. The constraint specifies the maximum allowable time between the first and last events constituting an occurrence of the episode. The following steps are to be followed to mine parallel episodes using this algorithm.

1. Load the data sequence using '**Event Sequence Loader**' or generate a data sequence using an appropriate model in '**Simulators**'.
2. Choose the '**Non-overlapped count with episode expiry constraint(Parallel)**' algorithm and the corresponding candidate generation algorithm '**Apriori candidate generation(Parallel)**' gets automatically selected.
3. Uncheck the '**Allow repeated event types**' option to avoid mining of episodes which have repeated events.
4. Enter the values of '**Freq. Th. (1 node)**' which is the frequency threshold used for one node episodes and '**Freq. Th.**' which is the frequency threshold used for episodes with more than one node. (These values can be automatically set by clicking the '**Suggest Threshold**' button in '**Event Sequence Loader**'.
5. Enter the value of '**Th. decay factor**'. The frequency threshold gets decayed by this factor after every iteration from two node episode onwards. (The default value is 1)
6. Enter the value of '**Max. Episode size**'. It is the maximum size of the episode that will be mined. (The default value is 10)
7. Enter the value of '**Episode expiry**'. This specifies the maximum span of an episode.
8. Click the '**Start mining**' button to get the list of frequent episodes of different sizes.

Click the '**Save episodes**' button to save the list of frequent episodes into a file. Click the '**Clear Episodes**' button to clear the list of frequent episodes. Uncheck the '**Show all cyclic permutations**' option to avoid listing those episodes which are cyclic permutation of another frequent episode.

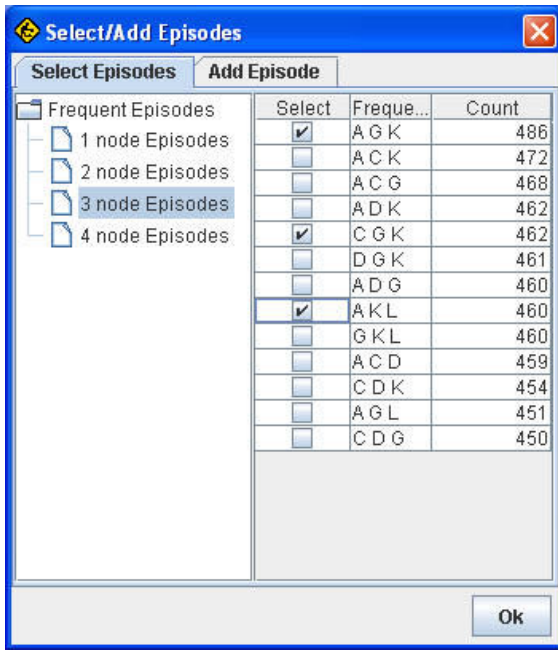


Episode Visualization

This tab allows the user to see the data stream. The raster plot of data (event types versus time of occurrence) is displayed under this tab. Each event type is identified with a number. User can move along the data by using the '<' and '>' tabs provided at the bottom of the plot. Using the '+' and '-' tabs here one can zoom in or zoom out the plot. By clicking the button '...' the legend (giving the numbers assigned to different event types) can be seen. This visualization tool also helps to see occurrences of specific episodes in the data as explained below.

To visualize occurrences of episodes:

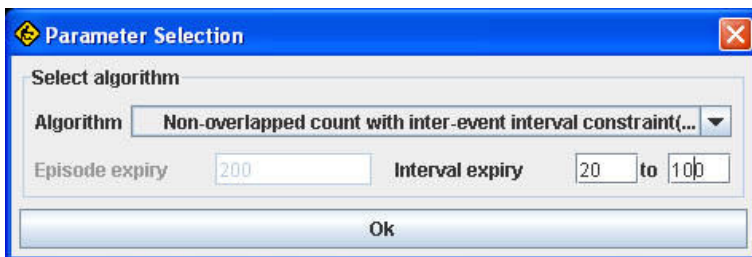
1. Click the '**Add/Select Episode**' button.
2. In the new window that appears select the '**Select Episodes**' tab (shown below) to visualize any of the mined frequent episodes. (This has a list of all frequent episodes).
3. Select the frequent episodes you wish to visualize.



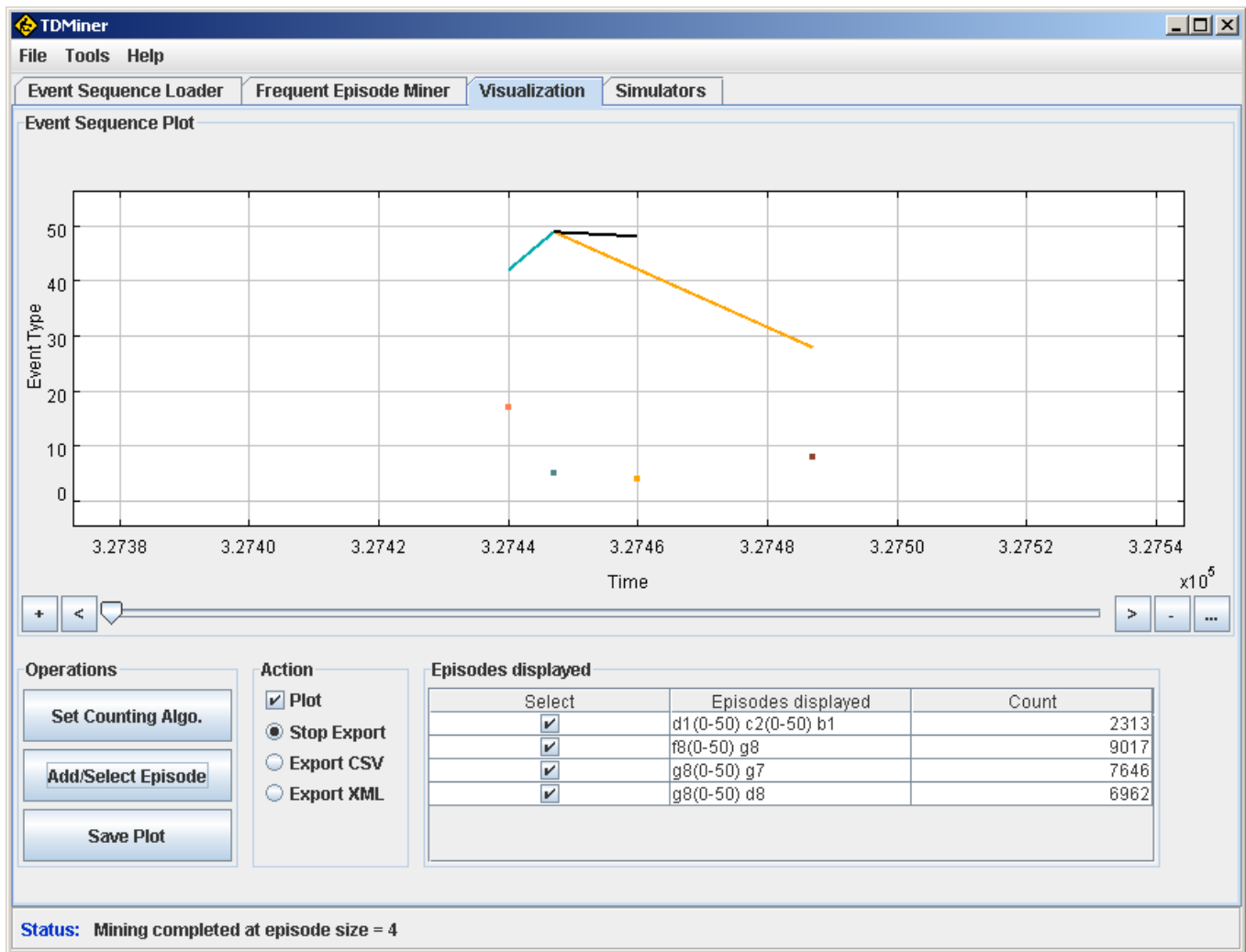
4. To visualize an episode of your choice select the '**Add Episode**' tab (shown below) in the new window.
5. Enter the episode in the format shown below.



6. Each occurrence of the episode is shown as a line drawn by connecting the events in the episode.
7. Click the '**Set Counting Algorithm**' button to change the frequency counting algorithm. A new window to set the mining algorithm and the parameters associated with it appears as shown below.



8. The plot can be saved using the '**Save Plot**' button.



Other Actions

The **Action** panel also provides for exporting the episode detail to files. By default the **Stop Export** button is selected. The episode detail may be exported as either comma separated values...

```
0,36.0,32.0
0,24.0,24.0
0,36.0,32.0
0,40.0,32.0
0,16.0,48.0
0,20.0,16.0
...
```

...or XML...

```

<?xml version="1.0"?>
<episode>
  <n>g8-g7</n>
  [7.0]
  <n>g8-g7</n>
  [8.0]
  <n>g8-g7</n>
  [8.0]
  <n>g8-d8</n>
  [32.0]
  <n>g8-g7</n>
  [8.0]
  <n>g8-d8</n>
  [32.0]
  <n>g8-g7</n>
  ...

```

The export occurs following the **Add/Select Episode** phase. When the **OK** button is selected the user will be prompted to select an appropriate file for the output.

Episode Simulation

Synthetic data streams with embedded frequent episodes in them can be generated using these simulators. The four models available for data generation are **Simulator Model I**, **Simulator Model II**, **Poisson Model** and **Data Generator**. The first three models are to generate data that resembles multi-neural spike data. The fourth model is a generic data synthesizer, which generates data having frequent episodes (with temporal constraints) embedded in randomly generated data.

Framework of Simulator Model I, Simulator Model II and Poisson Model :

These models that simulate multi-neural spike data are realized using the same basic framework consisting of a set of interconnected neurons. Each neuron in the network receives its inputs from a set of neurons called its input neurons and is connected to a set of receiver neurons. Each connection is assigned a weight. If the weight is positive the connection is said to be excitatory otherwise it is inhibitory.

Whether or not a neuron fires depends upon the total weighted input received by it. When a neuron fires, its output reaches all neurons into whose input terminals this neuron's output is connected. The effect of this neuron on others depends upon the nature of connection between them. For example, if the connection is excitatory the receiver neuron may be triggered to fire, while if it is inhibitory the receiver may be inhibited from firing. The firing neuron turns on for a random duration. After the completion of this period, the neuron is turned off for another random period. During this period it cannot be triggered to fire. This simulates the refractory period of real neuron cells.

The neurons are interconnected in two stages. First, random connections are established in the network whereby every pair of neurons is assigned a random connection weight uniformly distributed over $[-1, 1]$. Then the causative chain of neurons is embedded in the network by placing a high excitatory feed forward connection between the neurons involved in the chain. For example, if $(A \rightarrow B \rightarrow C)$ is the causative chain to be implemented in the network, then neuron A is connected to neuron B and a high positive weight is assigned to the connection from neuron A to neuron B . Similarly neuron B is connected to neuron C .

In order to start spiking activity in the network, a few neurons are randomly turned on. Each time a neuron fires (i.e. turns on), the label associated with the neuron and the time of occurrence are recorded as a pair in the output event stream.

Simulator Model I:

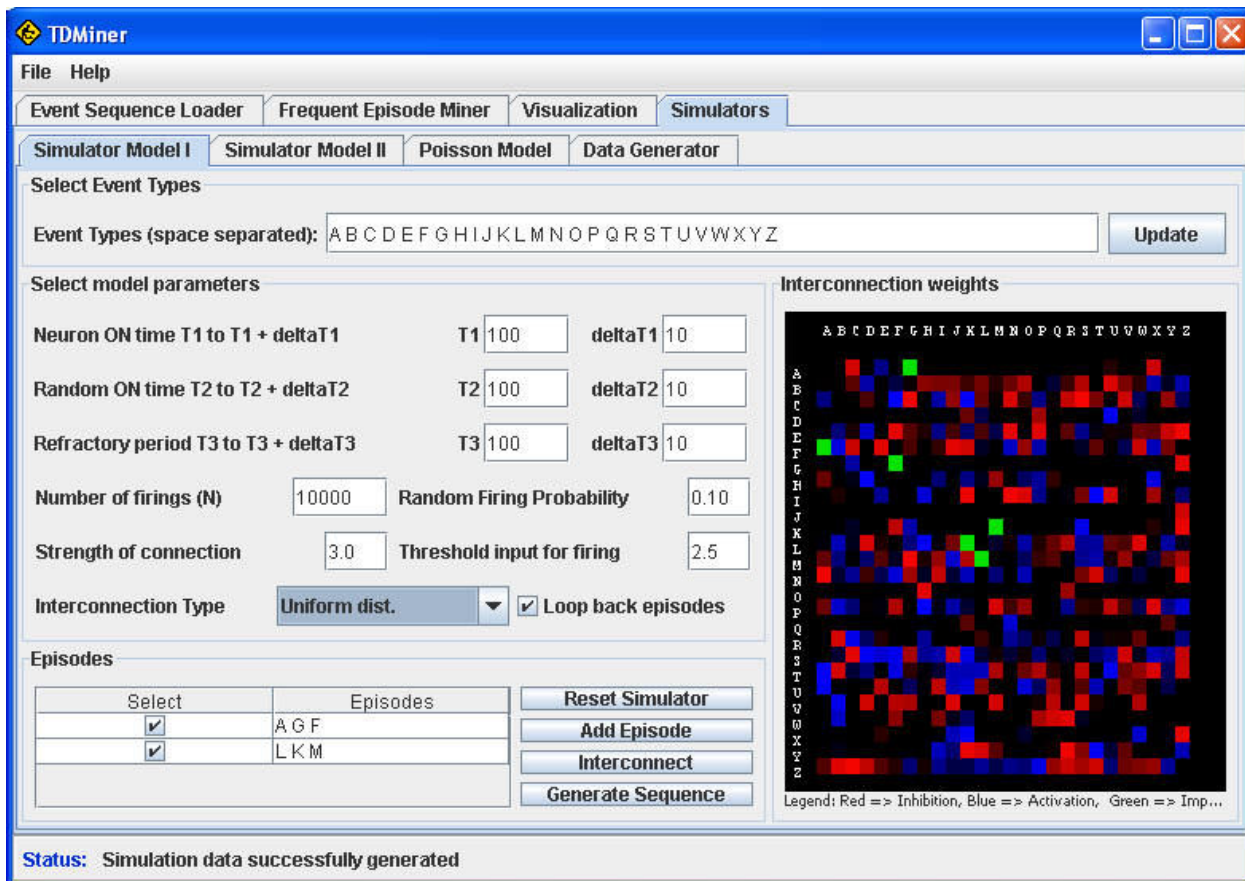
Description:

In this simulation model one neuron is randomly chosen at every step. The net input into this neuron is determined from the neurons feeding into it. Only those neurons that are already *on* will contribute to the input. If the net input exceeds a

threshold then the neuron is fired (and the neuron label and time are written into the output sequence). Even if the net input does not exceed the threshold, the neuron is fired with some probability called the random firing probability. The model needs the specification of some parameter values. Each time a neuron fires because there is sufficient input, it is kept ON for a time randomly selected from the time interval $[T1, T1 + \Delta T1]$. When a neuron fires randomly (without sufficient input), it is kept ON for a time drawn uniformly from $[T2, T2 + \Delta T2]$. The refractory period is a time drawn uniformly from the interval $[T3, T3 + \Delta T3]$. In addition to these times the user has to specify the Random firing probability, the threshold input for a neuron to fire and the required data length in terms of the total number of firings. The user specifies the episodes to be embedded also. As said earlier, the connection weights corresponding to the embedded patterns are kept at a high value which is also a parameter to be specified by the user. (Most of the parameters have default values as displayed). After interconnecting the network, the interconnection matrix is shown as a color coded picture. The user can then change any connection weight by going to the appropriate place in the image and right-clicking the mouse. (The specific connection that is clicked is shown in the bottom left of the display).

To generate data using Simulator Model I:

1. Enter the list of labels given to the neurons in the space given beside '**Event Types (space separated)**'.
2. Enter the value of **T1** and **$\Delta T1$** beside '**Neuron ON time T1 to T1 + $\Delta T1$** '. This specifies the random ON time of a selected neuron when its net input is greater than the specified threshold. The random ON time is uniform over the interval $[T1, T1 + \Delta T1]$.
3. Enter the value of **T2** and **$\Delta T2$** beside '**Random ON time T2 to T2 + $\Delta T2$** '. This specifies the random ON time of a selected neuron that fires even though its net input is less than the threshold.
4. Enter the value of '**Random firing Probability**'. This specifies the probability with which a selected neuron fires even when its net input is less than the threshold.
5. Enter the value of **T3** and **$\Delta T3$** beside '**Refractory period T3 to T3 + $\Delta T3$** '. This specifies the random period for which the neuron is turned off after firing (i.e. the refractory period).
6. Enter the value of '**Threshold input for firing**'. This specifies the threshold, which the net input of a neuron should exceed for it to fire.
7. Enter the '**Number of firings (N)**'. This specifies the total number of firings (i.e. the number of events in the generated data stream).
8. Click the '**Add Episode**' button to enter the episodes (in a new window) which will be embedded in the stream.
9. Enter the value of '**Strength of connection**'. This specifies the inter connection weight between neurons in the added episodes. For other neurons a random graph is first generated and its weight will be uniformly distributed between $[-1, 1]$.
10. Choose an '**Interconnection Type**' which selects the type of distribution to generate the initial graph (to determine how the pairs of neurons are chosen randomly). The distributions which are available are '**Uniform**' and '**Binomial**'. Another option '**Fully connected**' connects every neuron with every other neuron.
11. Check the '**Loop back episodes**' option to establish a strong connection even between the last neuron type of an episode with the first.
12. After all the above mentioned parameters are set, click the '**Interconnect**' button which generates the connection weights between the neurons. This is graphically shown at the right hand side. Here, red indicates an inhibitory connection and blue an excitatory connection, whereas green indicates the imposed value of connection.
13. Click the '**Generate Sequence**' button to generate the data sequence.
14. Click the '**Reset Simulator**' button to reset the simulator.
15. The generated data sequence can be saved as a text file by clicking the 'file' menu of the window.



Simulator Model II:

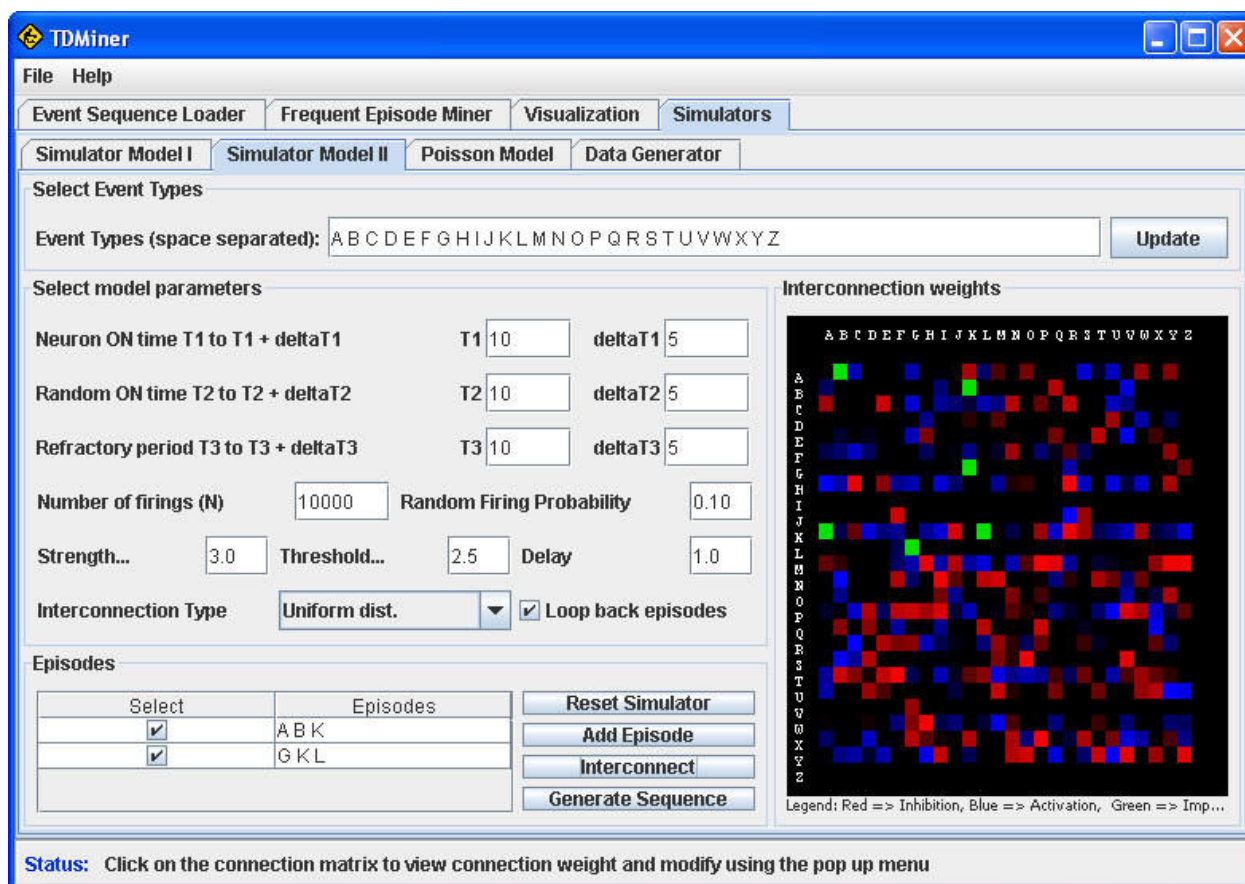
Description:

In the previous model, when a neuron fires the neurons to which this output is connected may not fire even if this input is sufficient for them. This is because we examine only one random neuron at each instant. Suppose there is a strong excitatory connection between A and B. When A fires, it is kept ON for some fixed time. Only if the neuron B happens to be picked up within this time, the firing of A would be propagated. In our simulation model II, we ensure this by running the simulator like a discrete event simulation system. Here all neurons to be examined are kept on a queue which is ordered according to the time at which neurons are to be examined. At each instant we select the next neuron from the queue. If the selected neuron fires all neurons that receive input from it are scheduled to be evaluated before the selected neuron turns off. This is done by posting these neurons on the queue with each neuron assigned a time that is randomly drawn from the interval over which the selected neuron would be ON. We can also choose to have a delay so that these neurons cannot fire till some time has elapsed. The action potential generated in one neuron takes finite time to reach its post-synaptic neurons and trigger an action potential in them. A delayed scheduling of the receiver neurons simulates the finite time required for the action potential to reach these units from the firing model. Whenever a neuron is picked from the queue, its total input is calculated and if it exceeds a threshold, it is fired. Even if the input is not sufficient, it is fired with some probability. In addition, for each instant, a randomly selected neuron is also kept on the queue so that a level of general activity is maintained in the network. This model also has all the parameters as in the earlier model. In addition, here we have to also specify the delay value.

To generate data using Simulator Model II

1. Enter the list of labels given to the neurons in the space given beside '**Event Types (space separated)**'.
2. Enter the value of **T1** and **deltaT1** beside '**Neuron ON time T1 to T1 + deltaT1**'. This specifies the random ON time of a selected neuron when its net input is greater than the specified threshold. The random ON time period is chosen from the interval $[T1, T1 + \text{deltaT1}]$.
3. Enter the value of **T2** and **deltaT2** beside '**Random ON time T2 to T2 + deltaT2**'. This specifies the random ON time of a selected neuron which fires even though its net input is less than the threshold.

4. Enter the value of '**Random firing Probability**'. This specifies the probability with which a selected neuron fires even when its net input is less than the threshold.
5. Enter the value of **T3** and **deltaT3** beside '**Refractory period T3 to T3 + deltaT3**'. This specifies the random period when the neuron is turned off after firing (refractory period).
6. Enter the value of '**Threshold input for firing**'. This specifies the threshold, which the net input of a neuron should exceed for it to fire.
7. Enter the '**Number of firings (N)**'. This specifies the total number of firings (i.e. the number of events in the generated data stream).
8. Click the '**Add Episode**' button to enter the episodes (in a new window) which will be embedded in the stream.
9. Enter the value of '**Strength of connection**'. This specifies the inter connection weight between neurons in the added episodes. For other neurons a random graph is first generated and its weight will be uniformly distributed between [-1 1].
10. Choose an '**Interconnection Type**' which selects the type of distribution to generate the initial graph (to determine how the pairs of neurons are chosen randomly). The distributions which are available are '**Uniform**' and '**Binomial**'. Another option '**Fully connected**' connects every neuron with every other neuron.
11. Check the '**Loop back episodes**' option to establish a strong connection even between the last neuron type of an episode with the first.
12. Enter the value of '**Delay**'. This specifies the minimum time before which a receiver neuron is not scheduled in the priority queue. This simulates the finite time required for the action potential to reach the receiver neurons from the firing neuron.
13. After all the above mentioned parameters are set, click the '**Interconnect**' button which generates the connection weights between the neurons. This is graphically shown at the right hand side. Here red indicates an inhibitory connection and blue an excitatory connection, whereas green indicates the imposed value of connection.
14. Click the '**Generate Sequence**' button to generate the data sequence.
15. Click the '**Reset Simulator**' button to reset the simulator.
16. We can save the generated data as a text file by clicking the 'file' menu of the window.



Poisson Model:

Description:

In this model, the activity of a neuron is given by the rate at which it is firing rather than by a single spike. The spike train of each neuron is treated as a non-homogeneous Poisson process. The rate parameter of the Poisson process associated with a neuron is a function of past firings of neurons connected to it. A induces an increase in the firing rate of B . When B starts firing with an increased rate it effectively inhibits the firing of A and initiates the activity of C . This is how activity propagates down the chain of imposed patterns. This is implemented as follows

$$N_j(t) = \sum_{i=0}^N (w_{ij} n_i(t - \Delta T, t))$$

$$\lambda_j(t) = \lambda_{\max} / (1 + \exp(-\Delta\lambda * N_j(t)))$$

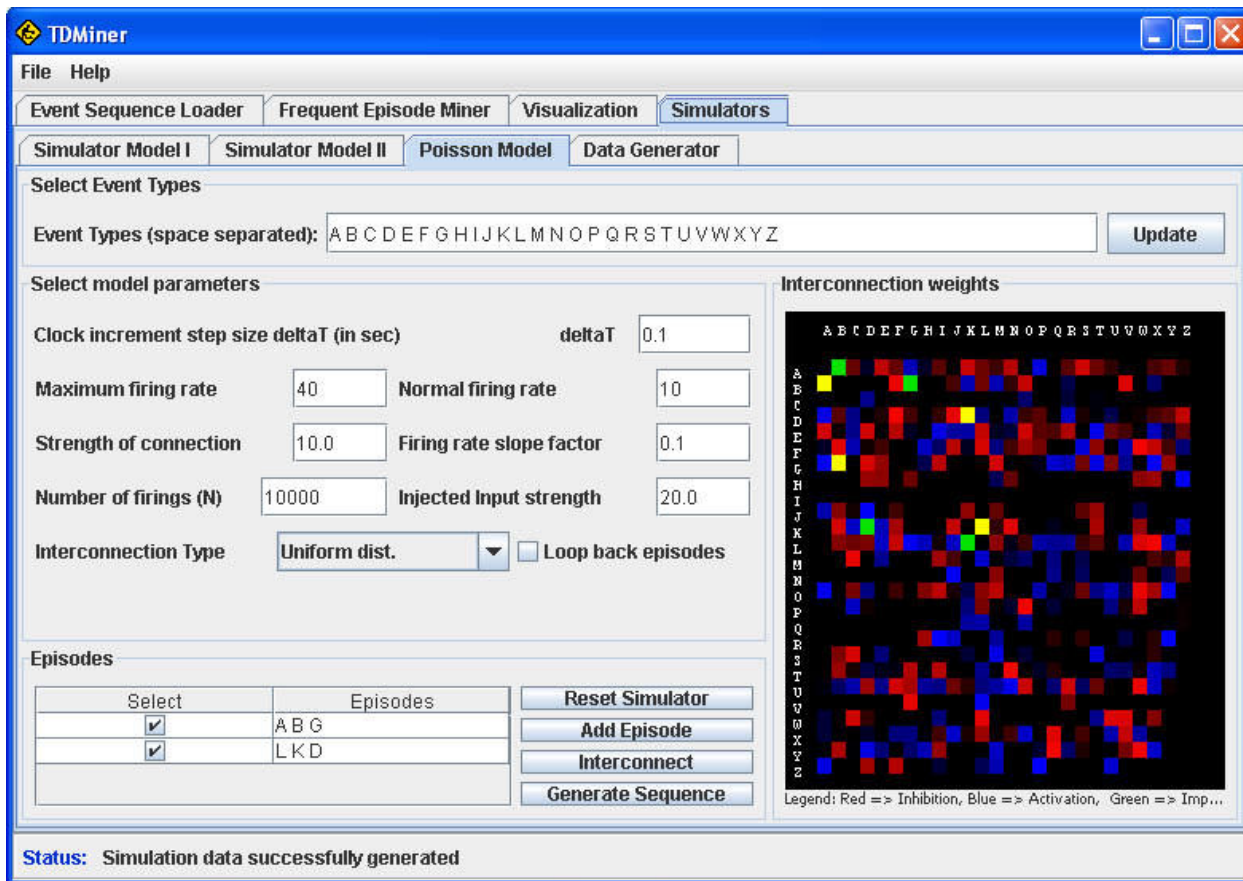
In the above equations $N_j(t)$ is the total input received by the j^{th} neuron. $n_i(t, t')$ is the number of spikes fired by the i^{th} neuron in time interval (t, t') . w_{ij} is weight of the connection $(i \rightarrow j)$. The firing rate of the j^{th} neuron at time t is given by $\lambda_j(t)$. λ_{\max} is the maximum firing rate and $\Delta\lambda$ controls the slope of logistic function.

The non-homogeneous Poisson process is simulated in time steps of ΔT . At each time step the firing rate of each neuron ($\lambda_j(t)$) is evaluated. The spike occurrence times are determined by generating a set of exponential random numbers such that the sum is within ΔT . The number of exponential random numbers generated is the number of spikes for the i^{th} neuron in the next interval ΔT .

A pattern is introduced in the network as follows. Suppose the imposed pattern is $A \rightarrow B \rightarrow C$. Then the connection $A \rightarrow B$ is made excitatory and the connection $B \rightarrow A$ is made inhibitory, and so on. Thus an increased firing rate of A induces increase in firing rate of B . When B starts firing with an increased rate it effectively inhibits firing of A and initiates activity of C . This is how activity propagates down the chain of imposed pattern.

To generate data using Poisson Model:

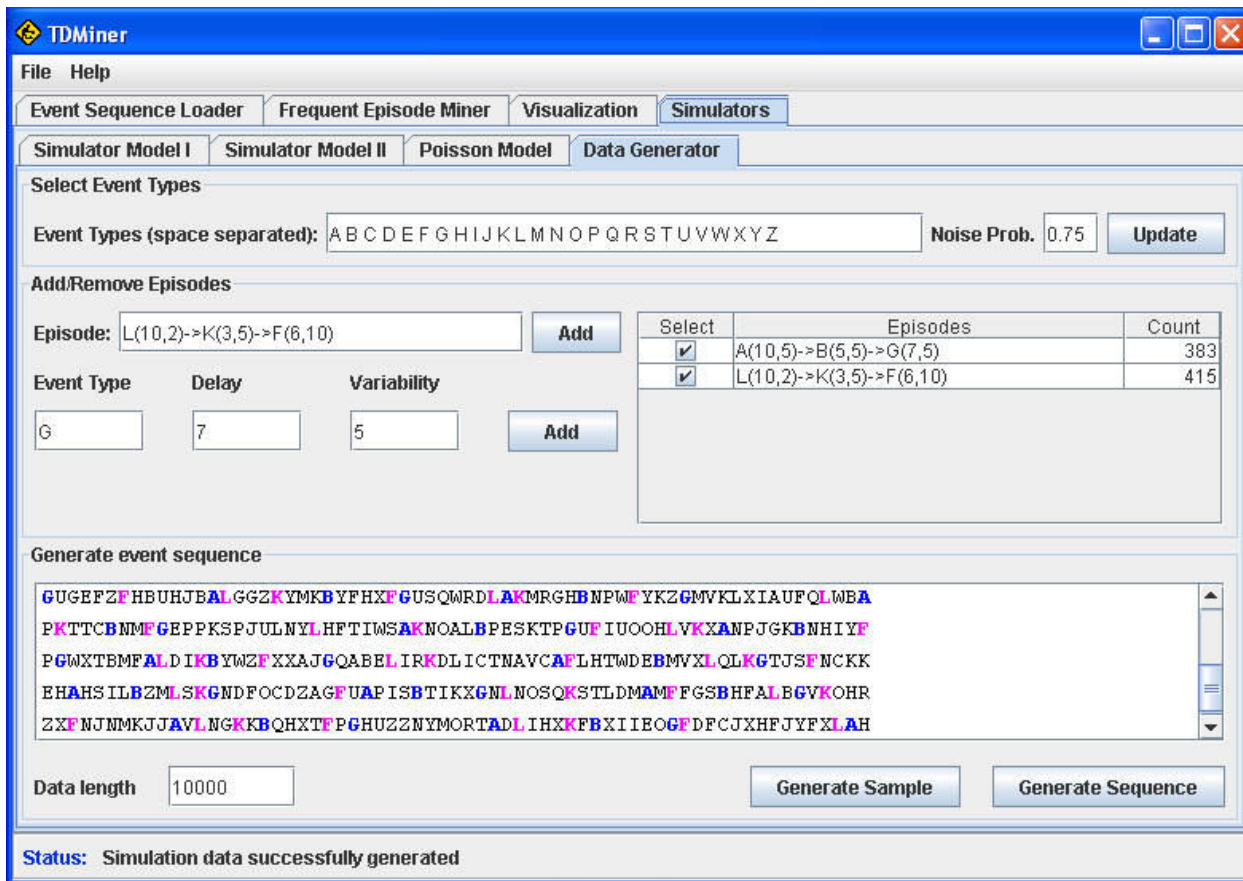
1. Enter the list of labels given to the neurons in the space given beside '**Event Types (space separated)**'.
2. Enter the value of '**deltaT**'. The non-homogeneous Poisson process is simulated in time steps of ΔT .
3. Enter the value of '**Maximum firing rate**' of neurons (λ_{\max}).
4. Enter the value of '**Firing rate slope factor**' ($\Delta\lambda$).
5. Enter the value of '**Normal Firing Rate**'. This specifies the firing rate for neurons when not part of any embedded chain.
6. Enter the '**Number of firings (N)**'. This specifies the total number of firings (i.e. the number of events in the generated data stream).
7. Click the '**Add Episode**' button to enter the episodes (in a new window) which will be embedded in the stream.
8. Enter the value of '**Strength of connection**'. This specifies the inter connection weight between neurons in the added episodes. For other neurons a random graph is first generated and its weight will be uniformly distributed between $[-1, 1]$.
9. Choose an '**Interconnection Type**' which selects the type of distribution to generate the initial graph (to determine how the pairs of neurons are chosen randomly). The distributions which are available are '**Uniform**' and '**Binomial**'. Another option '**Fully connected**' connects every neuron with every other neuron.
10. Check the '**Loop back episodes**' option to establish a strong connection even between the last neuron type of an episode with the first.
11. After all the above mentioned parameters are set, click the '**Interconnect**' button which generates the connection weights between the neurons. This is graphically shown at the right hand side. Here red indicates an inhibitory connection and blue an excitatory connection, whereas green indicates the imposed value of connection.
12. Click the '**Generate Sequence**' button to generate the data sequence.
13. Click the '**Reset Simulator**' button to reset the simulator.



Data Generator:

This is meant for generic synthetic data generation (i.e. not specific to model multi-neural spike data) synthetic. A pattern to be embedded is specified by a set of event types and the inter-event times.

1. Enter the list of labels given to the neurons in the space given beside '**Event Types (space separated)**'.
2. Enter the episodes that have to be embedded in the space given beside '**Episode**' using the following format $A(T_1, \Delta T_1) \rightarrow B(T_2, \Delta T_2) \rightarrow C(T_3, \Delta T_3)$. Here B occurs in the interval $[T_2, T_2 + \Delta T_2]$ after A has occurred. Similarly C occurs after B and A after C within the respective specified intervals.
3. Instead of entering the episode directly in the above mentioned format the episode can be constructed by entering the '**Event Type**', '**Delay**' (T) and '**Variability**' (ΔT) and adding the event by clicking the '**Add**' button.
4. Click the '**Add**' button beside '**Episode**' to add the episode to the episode list.
5. Enter the value of '**Noise Prob.**'. This is the probability with which a random event occurs in data stream if no event of an episode is scheduled at the present clock.
6. Enter the value of '**Data Length**' which specifies the length of the data sequence.
7. Click the '**Generate Sample**' button to generate a sample sequence. The episodes will be highlighted in the event sequence that is shown in the text box.
8. Click the '**Generate Sequence**' button to load the generated data stream.



Other Environments

GMiner: Matlab

Auxiliary Tools

Tools

Event Data Stream

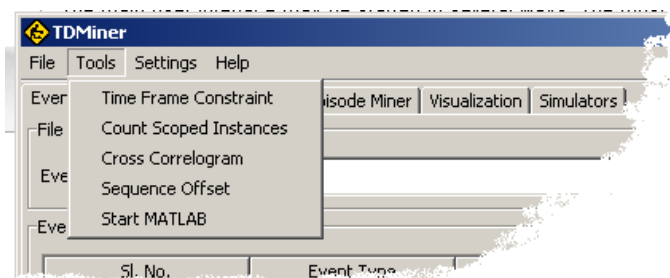
When dealing with large data streams or non-uniform data streams, it is desirable to restrict the action to a portion or portions of the total data stream. There are several reasons why restricting the data stream may be useful.

Loading

The total data stream may be simply too large for the memory to accommodate. In this case, it may be useful to restrict the data loaded in a variety of ways. It may be sufficient to load the data from a single span of time. It may be necessary to load data from regularly spaced time spans if the phenomenon suspected to be periodic or quasi-periodic. In addition to loading parts of a data stream, it may also be helpful to unload portions of the data stream as well, once they have served their purpose.

Prospecting

Clearly, eliminating irrelevant events from consideration would be beneficial in both processing effectiveness and less



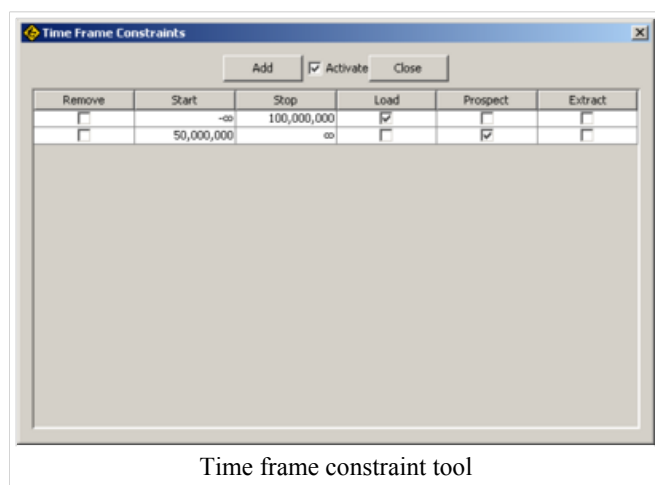
distraction from noise. During the discovery process judicious use of time frames can speed searches and focus the discovery to suspicious areas of the data stream. This is particularly useful when long episodes are expected.

Extraction

When making plots, sometimes there are just too many events on the screen to able to see anything.

Usage

The tool is found under the **Tool** menu titled **Time Frame Constraint**.



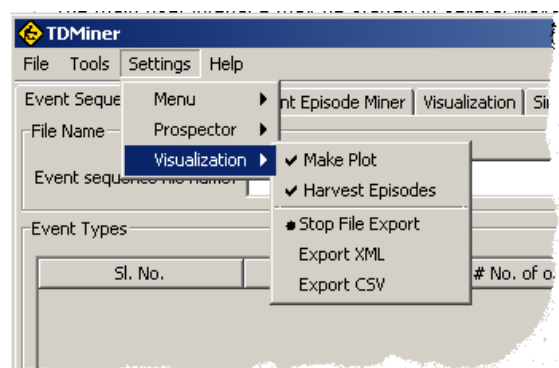
The dialog presents a list of time intervals, initially empty. Additional time constraints are introduced with the **Add** button. The time constraints are unordered and are applied during the various activities. The aggregate time constraint for an activity is the union of all the marked intervals. If a time frame is marked for **Prospect** but it has not been **Loaded** it will be treated as if it had been loaded. That is it will appear that no events occurred during that time. In other words, while frequencies of instances per event will be computed correctly any frequency per time may be distorted. Changes to the time constraints may be immediate or deferred depending on the status of the **Activate** checkbox. When the checkbox is marked as activated then changes happen as they are made. Otherwise, the changes are deferred until it is checked. This may be useful if a discovery process is running and you do not wish to disrupt it by changing the

time constraint while it is running. GMiner: Episode instance statistics

Settings

GMiner: Exporting files GMiner: Save event stream GMiner: Save episode instances as events GMiner: Cross correlogram GMiner: Sequence offset

Retrieved from



"[http://neural-code.cs.vt.edu/mediawiki/index.php/GMiner: User's guide \(expanded\)](http://neural-code.cs.vt.edu/mediawiki/index.php/GMiner:_User%27s_guide_%28expanded%29)"

- This page was last modified 21:09, 6 February 2007.
- This page has been accessed 13 times.
- Privacy policy
- About TDMiner
- Disclaimers