Enhanced Fast Causal Network Inference over Event Streams

Saurav Acharya and Byung Suk Lee

Department of Computer Science, University of Vermont, Burlington, VT 05405, USA
{sacharya,bslee}@uvm.edu

Abstract. This paper addresses causal inference and modeling over event streams where data have high throughput, are unbounded, and may arrive out of order. The availability of large amount of data with these characteristics presents several new challenges related to causal modeling, such as the need for fast causal inference operations while ensuring consistent and valid results. There is no existing work specifically for such a streaming environment. We meet the challenges by introducing a time-centric causal inference strategy which leverages temporal precedence information to decrease the number of conditional independence tests required to establish the causalities between variables in a causal network. (Dependency and temporal precedence of cause over effect are the two properties of a causal relationship.) Moreover, we employ change-driven causal network inference to safely reduce the running time further. In this paper we present the Order-Aware Temporal Network Inference algorithm to model the temporal precedence relationships into a temporal network and then propose the Enhanced Fast Causal Network Inference algorithm for learning a causal network faster using the temporal network. Experiments using synthetic and real datasets demonstrate the efficacy of the proposed algorithms.

Keywords: Causal inference; Event streams; Temporal data

1 Introduction

In recent years, there has been a growing need for active systems that can perform causal inference in diverse applications such as health care, stock markets, user activity monitoring, smart electric grids, and network intrusion detection. These applications need to infer the cause of abnormal activities immediately from their event streams, where the event arrival may be in order (e.g., [1, 2]) or out of order (e.g., [3–7]) such that informed and timely preventive measures can be taken. As a case in point, consider a smart electric grid monitoring application. The failure of a component can cause cascading failures, effectively causing a massive blackout. The identification of such cause and effect components in a timely manner enables preventive measures in the case of failure of a cause component, thereby preventing blackouts.

Causal network, a directed acyclic graph where the parent of each node is its direct cause, has been popularly used to model causality [8–14]. There are two distinct types of algorithms for learning a causal network: score-based [8–11] and constraint-based [12–15]. Both types of algorithms are slow and, therefore, not suitable for event streams where prompt causal inference is required. Score-based algorithms perform a greedy search (usually hill climbing) to select a causal network with the highest score from a large number of possible networks. With an increase in the number of variables in the dataset, the number of possible networks grows exponentially, resulting in slow causal network inference. On the other hand, constraint-based algorithms (e.g., PC algorithm [14]) discover the causal structure via a large number of tests on conditional independence(CI). There can be no edge between two conditionally independent variables in the causal network (e.g., [16, 17]). Two variables X and Y are said to be conditionally independent given a condition set S if there is at least one variable in S such that X and Y are independent (e.g., [18, 19]). In a causal network of n variables, the condition set S consists of all possible 2^{n-2} combinations of the remaining $n-2$ variables, and therefore the computational complexity grows exponentially as the number of variables increases. So, the current techniques for causal inference are slow and not suitable for event streams which have a high data throughput and where the number of variables (i.e., event types) is large. Besides, these techniques perform the time-consuming causal inference computations every time a new batch of events arrives even though there may not be significant enough changes in the event stream statistic.

With these concerns, this paper describes a new time-centric causal modeling approach to speed up the causal network inference. Every causal relationship implies temporal precedence relationship (e.g., [20, 21]). So, the idea is to exploit temporal precedence information as an important clue to reducing the number of required CI tests and thus maintaining feasible computational complexity. Four strategies are employed utilizing this idea to achieve fewer computations of CI tests. First, since causality requires temporal precedence, we ignore the causality test for those nodes with no temporal precedence relationship between them. Second, in the CI test of an edge, we exclude those nodes from the condition set which do not have temporal precedence relationship with the nodes of the edge; this strategy reduces the size of the condition set which is a major cause of the exponential computational complexity. Third, we perform the CI tests for weaker edges (i.e., having lower temporal strength) earlier to reduce the size of the condition set of stronger edges, thereby reducing the overall number of CI tests. The rationale for this is that weaker edges are more likely to be eliminated than stronger edges [22]. Fourth, we perform the causal inference computations only if there is a significant enough change in the temporal precedence relationships, which is a necessary condition for a change to occur in the resulting causal relationships. Such a change detection strategy helps to avoid unnecessary causal inference computations, and therefore, saves time.

Due to the reliance on the temporal precedence relationships in an event stream, events arriving *out of order* can bring ambiguities in the resulting causal directions. For instance, the precedence relationships represented in an edge and its reversed edge in a temporal network, which models the temporal precedence relationships, may not be significantly different enough to determine the edge direction. Intuitively, an undirected edge can be used to signify such an ambiguity. Thus, we propose a mechanism to decide between directed and undirected edge in the temporal network in such cases. Note that the constraint-based algorithms like the PC algorithm naturally handle out-of-order event arrivals, as these algorithms do not depend on the temporal ordering of events, and so they can provide a suitable baseline to evaluate the handling of out-of-order events in our proposed method.

The main contributions of this paper are summarized as follows. First, it presents a temporal network structure to represent temporal precedence relationships between event types and proposes an algorithm, *Order-Aware Temporal Network Inference (OATNI)*, to construct a temporal network applicable in the streaming environment. Second, it introduces a time-centric causal modeling strategy and proposes an algorithm, *Enhanced Fast Causal Network Inference (EFCNI)*, to speed up the learning of causal network. Third, it empirically demonstrates the advantages of the proposed algorithm in terms of the accuracy and speed of learning the causal network by comparing it against two state-of-art algorithms, the PC algorithm (details in Sections 3.4) and the FCNI algorithm [23].

This paper contains the results of a comprehensive study extended from our earlier work [23]. The two algorithms in our prior work, the *Temporal Network Inference* (TNI) and the *Fast Causal Network Inference* (FCNI), are extended to the OATNI and the FECNI algorithms, respectively. Specifically, two major extensions have been made. First, the speed of the causal inference mechanism has been increased with two strategies. As the first strategy, the CI tests are performed in the increasing order of the temporal strengths of the edges in order to remove the most probable spurious edge as early as possible, which decreases the condition set size. As the second strategy, presumably unnecessary causal inference computations are avoided by determining whether the changes in temporal precedence information in the event stream are significant enough to warrant such computations. Second, the previous work made an assumption that the event stream is in order. In this paper, the support for fast causal modeling over an out-of-order event stream is added so that the temporal precedence relationships cannot be relied upon as they are. In addition to these two major extensions, the presentation has been extended throughout in many parts of the paper.

The rest of this paper is organized as follows. Section 2 reviews the existing work on causal network inference. Section 3 presents the basic concepts used in the paper. Section 4 and Section 5 propose the learning algorithms of temporal network (OATNI) and faster causal network (EFCNI), respectively. Section 6 evaluates the proposed EFNCI algorithm. Section 7 concludes the paper and suggests further research.

2 Related Work

As explained earlier, there are two main approaches for causal network inference.

The first approach, score-based [8–11], performs greedy search (usually hill climbing) over all possible network structures in order to find the network that best represents the data based on the highest score. This approach, however, has two problems. First, it is slow due to the exhaustive search for the best network structure. An increase in the number of variables in the dataset increases the computational complexity exponentially. Second, two or more network structures, called the equivalence classes [24], may represent the same probability distribution, and consequently the causal directions between nodes are quite random. There is no technique for alleviating these problems in a streaming environment. Thus, score-based algorithms are not suitable for streams.

The second approach, constraint-based [12–15], does not have the problem of equivalence classes. However, it is slow as it starts with a completely connected undirected graph and thus performs a large number of CI tests to remove the edges between conditionally independent nodes. The number of CI tests increases exponentially with the increase in the number of variables in the dataset. To alleviate this problem, some constraint-based algorithms start with a minimum spanning tree to reduce the initial size of condition sets. However, this idea trades the speed with the accuracy of the causal inference. The constraint-based algorithms include IC* [12], SGS [13], PC [14], and FCI algorithm [14]. The FCI algorithm focuses on the causal network discovery from the dataset with latent variables and selection bias, which is quite different from the scope of this paper. The PC algorithm is computationally more efficient than IC* and SGS. This is why we evaluate the proposed *EFCNI* algorithm by comparing it against the PC algorithm. Like the others, the PC algorithm starts with a completely connected undirected graph. To reduce the computational complexity, it performs CI tests in several steps. Each step produces a sparser graph than the earlier step, and consequently, the condition set decreases in the next step. However, the computational complexity is still $O(n^2 \cdot 2^{n-2})$. (The details are explained in Section 3.4.) Therefore, the current constraint-based algorithms are not suitable for fast causal inference over streams.

There have been a number of research works on handing out-of-order event streams [3–7]. To the best of our knowledge, however, there exists no work applicable to the causal network inference. (Thus, a new approach is needed, and our approach is to allow *undirected edges* in the temporal network.) Johnson et al. [3] propose an algorithm for regular expression matching on streams with out-of-order data, which is not related to causal inference. The works by Li et al. [4] and Liu et al. [5] discuss the problem of processing event pattern queries over event streams that may contain out-of-order data. Li et al. [6] present a new architecture for stream systems for out-of-order query processing whereas Wang and Yu [7] propose algorithms for generating and matching queries to raise accuracy and shorten the response time as much as possible over out-of-order events. None of these works is related to causal network inference.

3 Basic Concepts

This section presents some key concepts needed to understand the paper.

3.1 Event streams

An event stream in our work is a sequence of continuous and unbounded timestamped events. An event refers to any action that has an effect and is created by an event owner. One event can trigger another event in chain reactions. Each event instance belongs to one and only one event type which is a prototype for creating the instances. Two event instances are related to each other if they share common attributes such as event owner, location, and time. We call these attributes *common relational attributes (CRAs)*.

In this paper we denote an event type as E_j and an event instance as e_{ij}, where i indicates the *CRA* and j indicates the event type.

Example 1. Consider a diabetic patient monitoring system in a hospital. Each patient is uniquely identifiable, and each clinical test or measurement of each patient makes one event instance. For example, a patient is admitted to the hospital, has their blood pressure and glucose level measured, and takes medication over a period of time. This creates the instances of the above event types as a result. Typical event types from these actions include hypoglycemic-symptoms-exists, blood-glucose-measurement-decreased, increased, regular-insulin-dose-given, etc. Note that the patient ID is the CRA, as the events of the same patient are causally related.

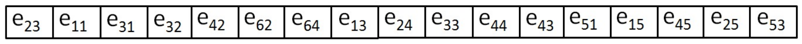
To facilitate the handling of events in a streaming environment, we use a time-based window over the stream. Typically, the application offers a natural observation period (e.g., hour) that makes a window. The causal relationship is only possible between events with the same CRA. Therefore, the events in a window are arranged by CRA and then ordered by the timestamp as they arrive, producing a *partitioned window* as a result. Figure 1 illustrates it. (We refer to the *partitioned window* simply as the *window* for the rest of the paper.)

With the arrival of a new batch of event instances, we augment each partition in the new window by prefixing it with the last instance of the partition with the same CRA value in the previous window. This is necessary in order to identify the related instances that are separated into the two consecutive batches.

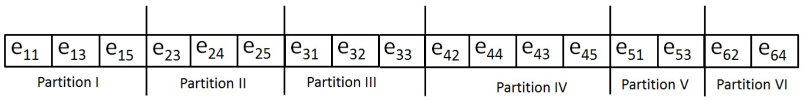
We support event streams which may be in order or out of order. An event stream is said to be in order if and only if every event in every partition arrives in the same temporal order as it was created. In other words, the stream is out of order if any event in any partition arrives in a different temporal order than it was created. The degree of out-of-order, d_{oo}, is given as

$$d_{oo} = \frac{\sum_{k=1}^{n_p} o_k}{n_{ins}} \tag{1}$$

where n_p is the number of partitions, o_k is the number of out-of-order events in the k-th partition and n_{ins} is the total number of events in all partitions. Note that d_{oo} is zero for in-order event streams.


e_{23} e_{11} e_{31} e_{32} e_{42} e_{62} e_{64} e_{13} e_{24} e_{33} e_{44} e_{43} e_{51} e_{15} e_{45} e_{25} e_{53}

(a) Events collected during an observation period (window).


e_{11} e_{13} e_{15} e_{23} e_{24} e_{25} e_{31} e_{32} e_{33} e_{42} e_{44} e_{43} e_{45} e_{51} e_{53} e_{62} e_{64}
Partition I
Partition II
Partition III
Partition IV
Partition V
Partition VI

(b) Events in the window partitioned by CRA.

Fig. 1. Partitioned window of events.

3.2 Causal networks

Causal network is a popularly used data structure for representing causality [8–11, 25]. It is a graph $G = (N, \xi)$ where N is the set of nodes (representing event types) and ξ is the set of edges between nodes. For each directed edge, the parent node denotes the cause, and the child node denotes the effect.

Consider the event stream of Figure 1. The causal relationships among the event types in the stream may be modeled as a causal network like the one shown in Figure 2.

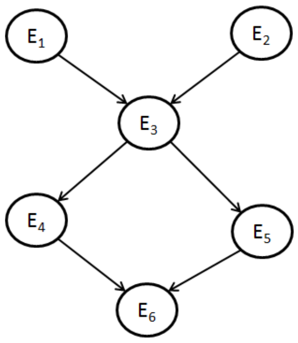


Fig. 2. Causal network.

The joint probability distribution of a set of n event types $\mathbf{E} \equiv \{E_1, ..., E_n\}$ in a causal network is specified as

$$P(\mathbf{E}) = \prod_{i=1}^{n} P(E_i|\mathbf{Pa_i})$$

where $\mathbf{Pa_i}$ is the set of the parent nodes of event type E_i.

3.3 Conditional mutual information

A popular approach for testing the conditional independence, with respect to the joint probability P, of two random variables X and Y given a subset S of random variables is conditional mutual information(CMI) (e.g., [15, 26]). CMI gives the strength of dependency between variables in a measurable quantity, which helps to identify strong and weak causal relationships in the final causal network.

To test whether X and Y are conditionally independent given S, we compute the conditional mutual information $I_{MI}(X, Y|S)$ as

$$I_{MI}(X,Y|S) = \sum_{x \in X} \sum_{y \in Y} \sum_{s \in S} p_{X,Y,S}(x,y,s) log_2 \frac{p_{X,Y|S}(x,y|s)}{p_{X|S}(x|s) p_{Y|S}(y|s)}$$

where p is the probability mass function calculated from the frequencies of variables.

We only keep the record of these frequencies, not the whole events, by updating them as a new batch of events arrives. Consequently, the independence test procedure is incremental in our case.

It is said that two variables X and Y are independent when $I_{MI}(X, Y|S) = 0$; otherwise, they are dependent. However, this presents us with the risk of spurious relationships due to weak dependencies (we cannot assume $I_{MI}(X, Y|S) = 10^{-5}$ and $I_{MI}(X, Y|S) = 10$ provide the same degree of confidence in the dependency). With an increase in the value of $I_{MI}(X, Y|S)$, the dependency between the variables X and Y grows stronger. Therefore, to prune out weak dependencies, we need to set a threshold value of mutual information below which we ignore the evidence as weak. To do so, we relate CMI with G^2 test statistics [14, 27] as below, where N_s is the sample size.

$$G^2(X, Y|S) = 2 \cdot N_s \cdot log_e 2 \cdot I_{MI}(X, Y|S)$$

G^2 follows the χ^2 distribution [28], with the degree of freedom *df* equal to $(r_x - 1)(r_y - 1) \prod_{s \in S} r_s$, where r_x, r_y, and r_s are the number of possible distinct values of X, Y, and S, respectively. So, we use χ^2 test, which provides a threshold based on *df* and significance level α, to validate the dependency result. We set α to the universally accepted value of 95%.

3.4 The PC algorithm

The key idea of the PC algorithm [14] is that a causal network has an edge between two nodes, X and Y, if and only if X and Y are not independent given any of the condition subsets of the remaining neighbor nodes [16, 17]. Algorithm 1 outlines the PC algorithm.

The algorithm has two parts. In the first part, the algorithm learns the topology of the causal network (Lines 1– 18). It starts with a completely connected undirected graph G of n nodes. Two sets are used for bookkeeping – *Neighbors(G,X)* and *SepSet(X,Y)*. *Neighbors(G,X)* gives the set of nodes adjacent to X, and *SepSet(X,Y)* gives the set of nodes which causes X and Y to be conditionally independent. Initially, *Neighbors(G,X)* has the remaining $n-1$ nodes and *SepSet(X,Y)* is empty. Then, the CI tests are performed between every pair of nodes that have an edge between them to determine whether they

Algorithm 1 PC algorithm

```
Require: Window W
 1: Construct the completely connected undirected graph $G$ of all nodes;
 2: for each node $X$ in $G$ do
 3:     Initialize Neighbors(G,X) as the set of nodes adjacent to $X$ in $G$;
 4: end for
 5: for each pair of nodes $X$ and $Y$ in $G$ do
 6:     Initialize an empty set SepSet(X,Y) as the set of nodes that causes independence
        between $X$ and $Y$ in $G$;
 7: end for
 8: $k \leftarrow 0$;
 9: repeat
10:     repeat
11:         Select any edge $X - Y$ such that $|Neighbors(G, X)\backslash Y| \geq k$;
12:         repeat
13:             Select any subset S of $Neighbors(G, X)\backslash Y$ such that $|Neighbors(G, X)\backslash Y|$
                $= k$;
14:             If X and Y are independent given S, remove $X - Y$ from G, remove Y from
                Neighbors(G,X), remove X from Neighbors(G,Y), and add S to SepSet(X,Y)
                and SepSet(Y,X);
15:         until every subset S of $Neighbors(G, X)\backslash Y$ such that $|Neighbors(G, X)\backslash Y|$
            $= k$ has been selected.
16:     until every edge $X - Y$ such that $|Neighbors(G, X)\backslash Y| \geq k$ has been selected.
17:     k = k + 1;
18: until no edge $X' - Y'$ satisfies $|Neighbors(G, X')\backslash Y'| \geq k$.
19: for each triplet of nodes X, Y, Z such that the edge $X - Y$ and $Y - Z$ exists in
    G but not $X - Z$ do
20:     Orient $X - Y - Z$ as X→Y←Z if and only if SepSet(X,Z) does not contain Y;
21: end for
22: repeat
23:     If there exists $X \rightarrow Y$ and $Y - Z$, but not $X - Z$, then orient $Y - Z$ as $Y \rightarrow Z$;
24:     If there exists $X - Y$ and a directed path from X to Y, then orient $X - Y$ as
        $X \rightarrow Y$;
25: until no edge can be oriented.
```

are conditionally independent of any other nodes in G. The edge between the conditionally independent nodes is removed. To ensure that all possible combination of nodes are considered in the condition set, the algorithm starts with an empty condition set and ends with the condition set with the maximum possible number of nodes. That is, in the algorithm k refers to the number of nodes in the condition set and is initially set to 0 to denote an empty condition set, and then k is gradually increased (by 1 at each iteration) and the CI tests are performed between every pair of nodes with the condition set of size k. This process is repeated until there is no edge left in G whose condition set size is greater than k. Eventually, an undirected network is obtained where an edge between two nodes denotes that these nodes are not conditionally independent in the presence of any of the other $n - 2$ nodes.

In the second part, the undirected network topology is assigned causal directions (Lines 19 – 25). It is done in three steps. First, if there are two edges $X - Y$ and $Y - Z$ but not $X - Z$ and *SepSet(X,Z)* does not contain Y, then $X - Y - Z$ is assigned the edge orientations $X \rightarrow Y \leftarrow Z$. The reason is that X and Z are dependent given Y, as the absence of Y in *SepSet(X,Z)* indicates that X and Z are not conditionally independent given Y [29, 30]. Second, if there are edges $X \rightarrow Y$ and $Y - Z$ but not $X - Z$, then $Y - Z$ is oriented as $Y \rightarrow Z$. The absence of an edge between X and Z means that X and Z are not dependent. A directed edge from X to Y and Y to Z with no edge between X and Z makes X and Z independent of Y [29, 30]. Thus, the edge between Y and Z is oriented as $Y \rightarrow Z$. Third, if there are edges $X - Y$ and a directed path from X to Y through any number of nodes, then $X - Y$ is oriented as $X \rightarrow Y$. This is necessary to make the graph acyclic, as the edge direction $X \leftarrow Y$ would make the graph cyclic.

4 Learning Temporal Precedence Relationships

In this section, we describe an incremental approach to model temporal precedence relationships from time-stamped events into a *temporal network*.

4.1 Temporal network model

A temporal network is a network of nodes representing event types where an edge between two nodes represents the temporal precedence relationship between them. To determine when an edge should be added in a temporal network, a measure providing an evidence of temporal precedence between the event types should be defined. The evidence we use is the frequency of the observation of an instance of E_j following an instance of E_i. We call this the *precedence frequency.*

Definition 1 (Precedence frequency). *The precedence frequency f_{ij} between two event types E_i and E_j is the total number of observations in which an event of type E_i precedes an event of type E_j over all partitions in the partitioned window.*

$$f_{ij} = \sum_{k=1}^{n_p} n_{(e_{ki}, e_{kj})}$$

where n_p is the number of partitions and $n_{(e_{ki}, e_{kj})}$ is the number of observations in which an event of type E_i precedes an event of type E_j in the k-th partition.
□

In our prior work [23], we assume that the event stream is in order and the temporal network from such an event stream is directed and acyclic. However, in an event stream with out-of-order event arrivals, the reliance on the temporal order for a definite temporal edge direction between event types may lead to ambiguous scenarios. For instance, the precedence frequencies between two event types E_i and E_j of an edge $E_i \rightarrow E_j$ and its reversed edge $E_i \leftarrow E_j$ may not differ significantly enough to determine an edge direction between them. An undirected edge $E_i - E_j$ is warranted to reflect such ambiguity. Thus, we support undirected edges as well as directed edges in the temporal network model. A directed edge between two nodes reflects a strong temporal precedence relationship between them, whereas an undirected edge reflects an ambiguous temporal precedence relationship between them. A threshold called the *temporal confidence* (θ) is used to select between directed and undirected edges, as presented in Rule 1 below.

Rule 1. Temporal edge direction selection
Suppose the precedence frequencies of an edge $E_i \rightarrow E_j$ and its reversed edge $E_i \leftarrow E_i$ are f_{ij} and f_{ji} such that either $f_{ij} > 0$ or $f_{ji} > 0$, respectively. Then, the edge direction between these two event types E_i and E_j (i.e., $\Xi(E_i, E_j)$) is selected as follows.

$$\Xi(E_i, E_j) = \begin{cases} E_i \rightarrow E_j & \text{if} \frac{f_{ij}-f_{ji}}{f_{ij}+f_{ji}} > \theta \\ E_i \leftarrow E_j & \text{if} \frac{f_{ji}-f_{ij}}{f_{ij}+f_{ji}} > \theta \\ E_i - E_j & \text{if} |\frac{f_{ij}-f_{ji}}{f_{ij}+f_{ji}}| \leq \theta \end{cases}$$

□

Given a temporal network, we define the edge strength, called the *temporal strength*, as follows.

Definition 2 (Temporal strength). *Consider an edge $E_i \rightarrow E_j$ ($i \neq j$) in a temporal network of n event types. Let f_{ij} be the precedence frequency from the event type E_i to the event type E_j. Then, we define the* temporal strength*, s_{ij}, of the edge $E_i \rightarrow E_j$ as*

$$s_{ij} \triangleq \frac{f_{ij}}{\sum_{k=1}^{n} f_{ik}}$$

□

That is, the temporal strength of $E_i \rightarrow E_j$ is the precedence frequency of (E_i, Ej) relative to the total precedence frequency over all children nodes of E_i.

4.2 Order-aware temporal network inference algorithm

The idea behind the *OATNI* algorithm is to collect events from an event stream in a *window* and then use temporal precedence information from the sequence of event pairs in the window to construct a temporal network at the event type level. The overall algorithm is centered on a *frequency matrix*, which is initially empty (i.e., all zero elements) and updated with each new batch of events.

The algorithm has two steps for each window, as outlined in Algorithm 2.

1. Update the frequency matrix *FM* by observing the precedence relationships of event pairs in the partitioned window (Lines 3–15). An element f_{ij} in *FM* reflects the total number of times events of type E_i precede events of type E_j ($i \neq j$). Each time an event pair (e_{oi}, e_{oj}) is observed in the event stream such that e_{oi} precedes e_{oj}, increase the value of f_{ij} by 1.

Algorithm 2 Order-Aware Temporal Network Inference (OATNI)

```
Require: Edgeless network structure TN, Event stream(s) $S$, Temporal confidence $\theta$
 1: Initialize an empty frequency matrix $FM$, an empty strength matrix $SM$, two empty
    buffers $B_p$ and $B_c$ (used to store "parent" events and "child" events, respectively);
 2: for each window $W$ in $S$ do
 3:   for each partition $P$ (corresponding to CRA $a$) in $W$ do
 4:     for i = 1 to $t_n - 1$ where $t_n$ is the number of unique timestamps in $P$ do
 5:       Clear $B_p$ and $B_c$;
 6:       Insert all events with timestamp $t_i$ and $t_{i+1}$ into $B_p$ and $B_c$, respectively;
 7:       for each event instance $e_{ap}$ in $B_p$ do
 8:         for each event instances $e_{ac}$ in $B_c$ do
 9:           if type($e_{ac}$) $\neq$ type($e_{ap}$) {//There cannot be causal relationships be-
              tween events of the same type.} then
10:             Increase the frequency of element $f_{type(e_{ap}),type(e_{ac})}$ in $FM$ by 1;
11:           end if
12:         end for
13:       end for
14:     end for
15:   end for
16:   for each pair of elements $f_{ij}$ and $f_{ji}$ such that $f_{ij} > 0$ or $f_{ji} > 0$ in $FM$ do
17:     $s_{ij} \leftarrow 0$, $s_{ji} \leftarrow 0$;
18:     if $\frac{f_{ij}-f_{ji}}{f_{ij}+f_{ji}} > \theta$ then
19:       Add an edge $E_i \rightarrow E_j$ in $TN$ and set its strength to $s_{ij} = \frac{f_{ij}}{\sum_{k=1}^{n} f_{ik}}$;
20:     else if $\frac{f_{ji}-f_{ij}}{f_{ij}+f_{ji}} > \theta$ then
21:       Add an edge $E_i \leftarrow E_j$ in $TN$ and set its strength to $s_{ji} = \frac{f_{ji}}{\sum_{k=1}^{n} f_{jk}}$;
22:     else if $\frac{|f_{ij}-f_{ji}|}{f_{ij}+f_{ji}} \leq \theta$ then
23:       Add an edge $E_i - E_j$ in $TN$ and set the strengths to $s_{ij} = \frac{f_{ij}}{\sum_{k=1}^{n} f_{jk}}$ and
          $s_{ji} = \frac{f_{ji}}{\sum_{k=1}^{n} f_{jk}}$;
24:     end if
25:   end for
26: end for
```

2. Determine the edges of the temporal network using *FM* (Lines 16–25). For each pair of nodes, add an edge according to Rule 1. For a directed edge added, e.g., $E_i \rightarrow E_j$, calculate its temporal strength and store it in s_{ij} of SM. For an undirected edge added, e.g., $E_i - E_j$, calculate the temporal strengths of both directions and store them in s_{ij} and s_{ji} of SM.

Note that when the events arrive in order, the *OATNI* algorithm reduces to the *TNI* algorithm by setting the threshold θ to 0 so that Rule 1 always chooses between the first two cases.

5 Learning Causal Network in Reduced Time

In this section, we describe a new approach which reduces the number of CI tests needed to infer the causal structure, thereby speeding up the learning process. Specifically, we explain the key ideas employed and discuss the concrete algorithm. We then prove the correctness of the algorithm and analyze its computational complexity.

5.1 Key ideas

Given that the key approach is to exploit temporal precedence relationships to learn the causal network, there are a number of ideas employed to reduce the causal network construction time. We begin by proposing some preliminary lemmas.

Lemma 1. *A CI test between two event types with no temporal precedence relationship is unnecessary.*

Proof. Two event types can a have causal relationship only if they have a temporal precedence relationship. Therefore, it is not necessary to perform a CI test (for detecting causality) between two event types which are not temporally related. □

Lemma 2. *Event types which do not have temporal precedence relationships with either of the two event types being tested for causality are not needed in the condition set of the CI test.*

Proof. Consider two event types, E_i and E_j, tested for causality, and consider another event type E_k $(k \neq i, j)$. E_k can causally influence E_i (or E_j) only if E_k has a temporal precedence relationship with E_i (or E_k). Therefore, the CI tests between E_i and E_j can safely exclude from the condition set those event types (i.e., E_k) which are not temporally related to either of them. □

Based on Lemma 1, the CI tests are performed only for the edges in the temporal network. That is, not every possible edges are considered for the CI tests leading to a reduced number of CI tests. Moreover, since the size of the condition set contributes to the number of CI tests exponentially, we use Lemma 2 to reduce the condition set size by including only those event types which have temporal relationships, hence possibly causal relationships, to the event types being tested.

We employ another idea to speed up the network inference further, based on Lemma 3 below.

Lemma 3. *The number of CI tests performed in the causal network inference decreases if the CI tests between event types are performed in an increasing order of their temporal strength.*

Proof. Event types with weaker temporal strengths between them have higher likelihood of being conditionally independent than those with stronger temporal strengths. Therefore, if the CI tests are performed between event types with the lowest temporal strength first, then the initial causal network becomes sparser faster and, consequently, the condition sets for the CI tests between event types become smaller faster. This leads to the reduction in the total number of CI tests performed through the causal network inference. □

Evidently, the reduction in the number of CI tests brings the reduction of running time.

Further, we employ the idea of reducing the overhead of causal network inference by performing it only when there are significant enough changes in temporal precedence relationships in the event stream. The rationale for this is that the causal network tends to absorb changes in the temporal network until the changes are significant enough. We introduce the temporal *precedence probability* as the measure to normalize the precedence frequencies between event types. The changes in the precedence probabilities give a normalized measure of the changes that have occurred in the temporal network since the last batch of events in the stream.

Definition 3 (Precedence probability). *The precedence probability p_{ij} between two event types E_i and E_j is defined as the ratio of f_{ij} and the summation of all precedence frequencies.*

$$p_{ij} = \frac{f_{ij}}{\sum_{x=1}^{n} \sum_{y=1}^{n} f_{xy}}$$

□

Let PM@t_i and PM@t_{i+1} be the matrices representing the precedence probabilities at the timestamps t_i and t_{i+1}, respectively. Then, the measure of change in the precedence information, called *precedence change* (C_p), is calculated as follows.

$$C_p = \sum_{x=1}^{n} \sum_{y=1}^{n} |p_{xy}@t_{i+1} - p_{xy}@t_i|$$

where p_{xy} is the element at the position (x, y) (i.e., event types (E_x, E_y)) in PM. Given this change measure, we update the causal network only if the calculated C_p exceeds a certain threshold, called the *precedence change confidence* (δ).

5.2 Enhanced fast causal network inference algorithm

The algorithm has four steps, as outlined in Algorithm 3.

1. The first step (Line 1) learns a temporal network by running the *OATNI* algorithm. The temporal network, which can have both directed and undirected edges, is set as the initial causal network.
2. The second step (Line 2) checks if there has been a significant enough change in the temporal precedence statistic (i.e., C_p) in the event stream from the last observation period, and stops if not.
3. The third step (Line 3) sorts the edges of the initial causal network in the increasing order of their temporal strength.
4. The fourth step (Lines 4–24) constructs the final causal network by pruning out the edges between independent nodes. CI tests are performed on every edge between adjacent nodes in the initial causal network to verify dependency between them. Conditionally independent nodes are considered spurious, and hence the edge between them is removed.

Algorithm 3 Enhanced Fast Causal Network Inference (EFCNI)

```
Require: Window $W$, Precedence change confidence $\delta$, Edgeless causal network $G$.
 1: Run the OATNI algorithm and initialize $G = (N, \xi)$ with the learned temporal
    network; {$N$ and $\xi$ are the set of nodes and the set of edges, respectively.}
 2: Calculate $C_p$. If $C_p < \delta$, then exit; {Stop if there is no significant change in the
    event stream.}
 3: Sort the edges in $\xi$ in the increasing order of their temporal strength.
 4: for each edge $(E_i, E_j) \in \xi$ do
 5:   independent = $IsIndependent(E_i, E_j, \phi)$, where $\phi$ is the empty set;
      {$IsIndependent(E_i, E_j, S)$ calculates $I_{MI}(E_i, E_j|S)$ for CI test.}
 6:   if independent is true then
 7:     Remove $(E_i, E_j)$ from $\xi$;
 8:   end if
 9: end for
10: $k \leftarrow 0$;
11: repeat
12:   for each edge $(E_i, E_j) \in \xi$ do
13:     Construct a set of condition sets, $Z$, each of cardinality $k$ from the parents of
        $E_j$ excluding $E_i$;
14:     repeat
15:       Select any subset $S$ from $Z$;
16:       independent = $IsIndependent(E_i, E_j, S)$;
17:       Remove S from $Z$;
18:     until $Z$ is empty or independent is true
19:     if independent is true then
20:       Remove $(E_i, E_j)$ from $\xi$;
21:     end if
22:   end for
23:   $k = k + 1$;
24: until there is no $E_j$ in any edge $(E_i, E_j) \in \xi$ with $k$ incident edges.
```

The main difference from the PC algorithm is the manner in which the CI tests are performed. In the PC algorithm, the condition set S for an edge $E_i - E_j$ (*undirected*) considers the neighbors of both E_i and E_j whereas in the EFCNI algorithm, the condition set S for an edge $E_i \to E_j$ (*directed*) needs to consider only the parents of E_j. (E_j is independent of the parents of E_i that do not have edge to E_j.) Consequently, fewer CI tests are needed. In addition, note that the *EFCNI* algorithm reduces to the *FCNI* algorithm by omitting the second and the third steps.

5.3 Correctness of the algorithm

To prove the correctness of the algorithm, it suffices to prove the correctness of our approach which starts with a temporal network as the initial causal network and removes edges through CI tests on them. We show the correctness as follows. First, a temporal precedence relationship is a necessary condition for inferring causality [20]. Therefore, causal relationship subsumes temporal precedence relationship, that is, the causal network is a subgraph of the temporal network (Lemma 1). Second, a causal network should satisfy the *Causal Markov Condition (CMC)* [13, 16, 31] where for every node X in the set of nodes N, X is independent of its non-descendants excluding its parents (i.e., $N\backslash(Descendants(X)\cup Parents(X))$) given its parents. In a temporal network of vertex (or node) set N, a node is temporally independent, and therefore causally independent, of all its non-descendants (except its parents) given its parents (Lemma 2).

5.4 Complexity analysis

Given n nodes, the computational complexity of the EFCNI algorithm is $O(n^2 \cdot 2^{n-2})$ in the worst case and $O(n)$ in the best case.

Proof. The computational complexity of the EFCNI algorithm is governed by the total number of possible CI tests which is calculated by summing up the number of CI tests involving each edge. In the worst case, the number of edges in the network is that of a completely connected graph and all edges are undirected. The number of edges in a completely connected graph of n nodes is $\frac{n(n-1)}{2}$. For every edge between two nodes, the remaining $n-2$ nodes are considered in the condition set, as the graph is completely connected and undirected. Therefore, to test conditional independence between a pair of nodes in an edge, there are 2^{n-2} CI tests to perform. Consequently, the total number of CI tests for all edges is $\frac{n(n-1)}{2} \cdot 2^{n-2}$, resulting in the computational complexity of $O(n^2 \cdot 2^{n-2})$.

In the best case, the initial causal network (i.e., temporal network) is a directed linear graph and the number of edges is the minimum (i.e., $n-1$). In such a graph, there are $n-2$ edges with one incoming edge to either of the nodes and one edge with no incoming edge to either of the nodes. For the edges with one incoming edge, the condition set size is one, and therefore there are two CI tests to perform. For $n-2$ such edges, there are $2n-4$ CI tests. For the remaining one edge with no incoming edge to either of the nodes, there is only one CI test to perform. Therefore, there are $2n-3$ CI tests to perform in the best case, resulting in the computational complexity of $O(n)$. □

The computational complexity of the PC algorithm is $O(n)$ in the best case and $O(n^2 \cdot 2^{n-2})$ in the worst case [14]. Note that, while the computational complexities are the same, the EFCNI algorithm starts with a sparse network as the use of temporal precedence relationships removes many of the edges. So, it starts closer to the best case. In contrast, the PC algorithm always starts with a completely connected dense network. So, it starts from the worst case. **As a result, in practice the EFCNI algorithm shows significant improvement in runtime over the PC algorithm.**

The computational complexity of the FCNI algorithm is $O(n)$ in the best case and $O(n \cdot 2^{n-2})$ in the worst case [23]. FCNI's worst case computation complexity is lower than that of EFCNI by a factor of n. However, unlike the EFCNI algorithm, the FCNI algorithm is not suitable for out-of-order event streams.

Moreover, as mentioned earlier, the EFCNI algorithm reduces to the FCNI algorithm when the events are in order and θ is zero. **As a result, in practice the EFCNI algorithm is at least as fast and accurate as the FCNI algorithm when the events are in order and preserves the accuracy in the face of out-of-ordered events, compromising the runtime to some extent as an increasing number of events arrive out of order.**

6 Performance Evaluation

We conducted experiments to compare the proposed EFCNI algorithm against the FCNI and the PC algorithms. There are three sets of experiments – first in terms of the accuracies of the resulting causal networks, second the running time, and third the number of CI tests required. In each set of experiments, we consider both the cases of stream being in order and out of order and also see the effect of the EFCNI's change-driven causal network construction strategy by comparing it with FCNI and PC when there are changes in the event stream statistic. Section 6.1 describes the experiment setup, Section 6.2 explains the datasets used, and Section 6.3 presents the experiment results.

6.1 Experiment setup

6.1.1 Evaluation metrics. The evaluation metrics are the speed of the causal network generation and the accuracy of the generated causal network. The running time is the CPU time, and the number of performed CI tests affects the speed. The accuracy is evaluated by examining how closely the constructed causal network structure resembles the target causal network. For this, we adopt the *structural Hamming distance* proposed by Tsamardinos et al. [32] as the measure. The nodes (i.e., event types) are fixed as given to the algorithms, and therefore the network structures are compared with respect to the edges between nodes. There are three kinds of possible errors in the causal network construction: reversed edges, missing edges, and spurious edges. We use the number of erroneous edges of each kind as the evaluation metric.

6.1.2 Platform. The experiments are conducted on RedHat Enterprise Linux 5 operating system using Java(TM) 2 Runtime Environment–SE 1.5.0_07 in Vermont Advanced Computing Core (VACC) cluster computers.

6.2 Datasets

Experiments are conducted using both synthetic and real datasets.

Synthetic datasets. A synthetic dataset is reverse-engineered from a target causal network. Given control parameters in Table 1, the idea is to generate a random causal network, and then convert the causal network to an event stream which reflects the underlying probability distribution of the causal network. Specifically, there are three steps. First, N_{ET} nodes are created and edges are added randomly, and random conditional probabilities are assigned to each edge. Each node can have up to Max_{NC} edges from cause nodes and up to Max_{NE} edges to effect nodes. (We set both Max_{NC} and Max_{NE} to 3 for the experiments presented here.) Second, a joint probability distribution (JPD) table is built from the conditional probabilities assigned to edges of the target causal network. The rows of the JPD table collectively cover all event sequences possible, while each row has its own probability. Third, the probability for each row in the JPD table is multiplied by N_O to calculate the number of repetitions of that event sequence in the dataset. We assume that the event owner is the *CRA* for the dataset.

Parameter	**Meaning**
N_O	Number of event owners (with unique ID)
N_{ET}	Number of event types (i.e., nodes)
Max_{NC}	Maximum number of cause events (parents)
Max_{NE}	Maximum number of effect events (children)

Table 1. Control parameters for synthetic event stream generation.

The size of a JPD table grows exponentially with N_{ET} and therefore we use parallel processing for the event stream generation. The JPD table is divided into multiple partitions and the dataset is created by running parallel processes over each of these partitions. The dataset is thus represented by a collection of files in which the events are shuffled according to the owner ID while preserving the temporal order.

There are five cases of datasets, DS1 through DS5, according to the number of nodes in the represented target causal networks (see their profiles in Table 2). The target causal networks have 4, 8, 12, 16 and 20 nodes, respectively. They are created with 1, 2, 16, 64 and 512 parallel processes, respectively, thus consisting of 1, 2, 16, 64 and 512 files, respectively. Each row of a synthetic dataset represents one event instance. To obtain out-of-order event streams, each case of datasets is shuffled randomly up to the required degree of out-of-order (see Equation 1). Changes in the event stream statistic is achieved by altering the precedence frequencies of events. Specifically, we generate six batches of the event stream for six observation points (t_1 through t_6) with the C_p values of $14\%, 16\%, 4\%, 6\%, 10\%, and 12\%$, respectively, for each case of the datasets.

Real dataset. The real dataset D_R contains diabetes lab test results [33] of 70 different patients over a period ranging from a few weeks to a few months. The dataset has a total 28143 records, about 402 records for each patient. Each record has four fields – date, time, test code, test value. The clinical data of a patient is independent of other patients. Therefore, the patient ID is the *CRA* for this dataset. There are 20 different test codes appearing in the file (shown in

Dataset	$\mathbf{N}_{ET}$	$\mathbf{N_{edges}}$	$\mathbf{N_O}$	$\mathbf{N_{ins}}$
DS1	4	4	5000	15128
DS2	8	15	30000	124475
DS3	12	22	500000	3173246
DS4	16	39	6553600	50247293
DS5	20	49	52428800	510971687

(N_{edges} is the number of *actual* edges in the network. N_{ins} is the average number of event instances in the datasets of each case.)

Table 2. Profiles of the five synthetic datasets.

the left column of Table 3) from which we define event types of interest (shown in the right column of Table 3).

Test Code	**Event Type**
Regular insulin dose	Regular-insulin-dose-given(RIDG)
NPH insulin dose	NPH-insulin-dose-given(NIDG)
UltraLente insulin dose	UltraLente-insulin-dose-given(UIDG)
Unspecified BGM*	Blood-glucose-measurement-increased(BGMI) Blood-glucose-measurement-decreased(BGMD)
Pre-breakfast BGM*	
Post-breakfast BGM*	
Pre-lunch blood BGM*	
Post-lunch BGM*	
Pre-supper BGM*	
Post-supper BGM*	
Pre-snack BGM*	
Hypoglycemic symptoms	Hypoglycemic-symptoms-exist(HSE)
Typical meal ingestion	Typical-meal-ingested(TMI)
More than usual meal ingestion	More-than-usual-meal-ingested(MTUMI)
Less than usual meal ingestion	Less-than-usual-meal-ingested(LTUMI)
Typical exercise activity	Typical-exercise-taken(TET)
More than usual exercise activity	More-than-usual-exercise-taken(MTUET)
Less than usual exercise activity	Less-than-usual-exercise-taken(LTUET)

(Note BGM* : blood glucose measurement)

Table 3. Event types defined from the diabetes dataset.

6.3 Experiment results

We run the EFCNI, FCNI and PC algorithms over each type of the five synthetic datasets and the real dataset. We present our evaluation in each of the three sets of experiments. First, we evaluate the accuracy of the generated causal networks against the target causal network and determine how closely they resemble the true causal network. Specifically, we count the number of spurious edges, the number of missing edges, and the number of reversed edges. Second, we evaluate the running time (CPU time), and third, we evaluate the number of CI tests performed. We show that reducing the number of CI tests is the key to reducing the running time of causal network inference. In each set of experiments, the evaluation covers the scenarios of the event stream being in order and out of order, and, additionally, the scenario of the event stream statistic changing. For the latter scenario, the value of the precedence change confidence δ is set to 9% for all synthetic datasets (DS_1 through DS_5). For the experiments involving in-order event streams, the temporal precedence confidence θ is set to zero (so EFCNI reduces to FCNI) and, for the experiments involving out-of-order event streams, it is set to $24.80\%, 17.23\%, 18.19\%, 21.97\%, 26.50\%$ (each determined after training from 70% of the data) for all synthetic datasets. The experiment is repeated ten times for each dataset (DS_1 through DS_5 and D_R) to calculate the average.

6.3.1 Comparison of the accuracies of the PC, FCNI, and EFCNI algorithms

6.3.1.1 ***When the events arrive in order***

Table 4 presents the number of erroneous edges in the causal network produced by the PC, FCNI, and EFCNI algorithms. The results show that the accuracy of the causal network from the EFCNI algorithm is similar to that of the FCNI and PC algorithms. First, the number of missing and the number of spurious edges are comparable among all three algorithms. This is due to the reliance of the three algorithms on the same test statistics (CMI in our case) to infer the independence of two event types. Additionally, each number is the same between EFCNI and FCNI because EFCNI reduces to FCNI. Second, the number of reversed edges is zero for both the FCNI and EFCNI algorithms. Clearly the FCNI and EFCNI algorithms, through the temporal network, are much better at determining the correct causal edge directions. It is because of the fact that the cause always precedes its effect is embodied in the temporal precedence relationship. Overall, the results show that, when the event stream is in order, the EFCNI algorithm produces the same topology as the FCNI algorithm and almost the same topology as the PC algorithm, while the accuracy of the causal directions in the EFCNI algorithm remains the same as the FCNI algorithm and is improved over the PC algorithm.

6.3.1.2 ***When the events arrive out of order***

Table 5 presents the number of erroneous edges in the causal network produced by the three algorithms for varying degree of out-of-order in the event stream. We show the results for the two datasets DS_4 and DS_5 only; the results from the other datasets are consistent with the results from the two datasets. We make two observations from the results. First, the PC algorithm is more resilient to the out-of-order event arrival than the FCNI or EFCNI algorithm. **The number of spurious edges and the number of missing edges are higher in the**

Type of Erroneous Edges	Algorithm	Dataset					
		$\mathbf{DS_1}$	$\mathbf{DS_2}$	$\mathbf{DS_3}$	$\mathbf{DS_4}$	$\mathbf{DS_5}$	$\mathbf{D_R}$
Missing	*PC*	0	0	0	0	1	1
	FCNI	0	1	0	0	1	1
	EFCNI	0	1	0	0	1	1
Reversed	*PC*	0	2	0	2	3	2
	FCNI	0	0	0	0	0	0
	EFCNI	0	0	0	0	0	0
Spurious	*PC*	0	3	0	4	3	1
	FCNI	0	3	0	4	3	1
	EFCNI	0	3	0	4	3	1

Table 4. Number of erroneous edges in an in-order event stream.

EFCNI algorithm than in the PC algorithm when the degree of out-of-order is large (i.e., $d_{oo} = 20\%, 25\%$). The reason is that the PC algorithm does not depend on the temporal precedence order for causal network inference at all whereas FCNI and EFCNI do. Second, between the FCNI algorithm and the EFCNI algorithm, EFCNI results in a comparable number of erroneous edges as PC while FCNI results in a larger number of erroneous edges than EFCNI or PC. The FCNI algorithm completely depends on the temporal order of the events to generate the causal network structure and, consequently, is sensitive to even a small change in the order of the events. In contrast, the EFCNI algorithm employs the OATNI algorithm where the temporal confidence threshold mechanism selects *undirected* edges in the temporal network when the temporal precedence is ambiguous, and this mechanism makes EFCNI more resilient to the changes than FCNI.

Type of Erroneous Edges	Algorithm	$\mathbf{d_{oo}}$ for $\mathbf{DS_4}$						$\mathbf{d_{oo}}$ for $\mathbf{DS_5}$					
		0%	**5%**	**10%**	**15%**	**20%**	**25%**	**0%**	**5%**	**10%**	**15%**	**20%**	**25%**
Missing	*PC*	0	0	0	0	0	0	1	1	1	1	1	1
	FCNI	0	1	3	4	4	7	1	4	6	7	11	13
	EFCNI	0	0	0	0	1	2	1	1	1	1	1	2
Reversed	*PC*	2	2	2	2	2	2	3	3	3	3	3	3
	FCNI	0	2	3	5	8	14	0	3	5	6	9	15
	EFCNI	0	0	0	0	0	0	0	0	0	0	0	0
Spurious	*PC*	4	4	4	4	4	4	3	3	3	3	3	3
	FCNI	4	7	9	12	13	17	3	5	9	11	18	23
	EFCNI	4	4	4	4	4	6	3	3	3	3	4	7

Table 5. Number of erroneous edges in an out-of-order event stream for different degrees of out-of-order (d_{oo}) (datasets: DS_4 and DS_5).

6.3.1.3 ***When the event stream has changing temporal precedence statistic***

Table 6 presents the number of erroneous edges in the causal networks resulting from the three algorithms for the event stream with changing temporal precedence statistic. As expected, the number of erroneous edges from the FCNI or PC algorithm is not affected by these changes because the causal network inference is run every time a new batch of events arrives. In contrast, for the EFCNI algorithm, the number of erroneous edges increases when C_p is lower than δ (at t_3 and t_4). (Note that in such cases the causal network inference is not run.) Additionally, the errors are larger at t_4 than t_3. This is because the higher value of C_p results in a greater difference between the causal network constructed and the true causal network and, more importantly, because the accuracy of the resulting causal network keeps on degrading as we keep on skipping the causal inference. On the other hand, for a batch of events with C_p greater than δ (i.e., at t_1, t_2, t_5, and t_6), the EFCNI algorithm rebuilds the causal network and, consequently, the resulting causal network reflects the true causal network representing the event stream seen thus far. Therefore, at these time points the number of erroneous edges remains the same as if the event stream had no change.

Type of Erroneous Edges	**Algorithm**	**DS$_4$**						**DS$_5$**					
		$\mathbf{t_1}$	$\mathbf{t_2}$	$\mathbf{t_3}$	$\mathbf{t_4}$	$\mathbf{t_5}$	$\mathbf{t_6}$	$\mathbf{t_1}$	$\mathbf{t_2}$	$\mathbf{t_3}$	$\mathbf{t_4}$	$\mathbf{t_5}$	$\mathbf{t_6}$
Missing	*PC*	0	0	0	0	0	0	1	1	1	1	1	1
	FCNI	0	0	0	0	0	0	1	1	1	1	1	1
	EFCNI	0	0	1	2	0	0	1	1	2	4	1	1
Reversed	*PC*	2	2	2	2	2	2	3	3	3	3	3	3
	FCNI	0	0	0	0	0	0	0	0	0	0	0	0
	EFCNI	0	0	0	1	0	0	0	0	1	2	0	0
Spurious	*PC*	4	4	4	4	4	4	3	3	3	3	3	3
	FCNI	4	4	4	4	4	4	3	3	3	3	3	3
	EFCNI	4	4	5	7	4	4	3	3	5	8	3	3

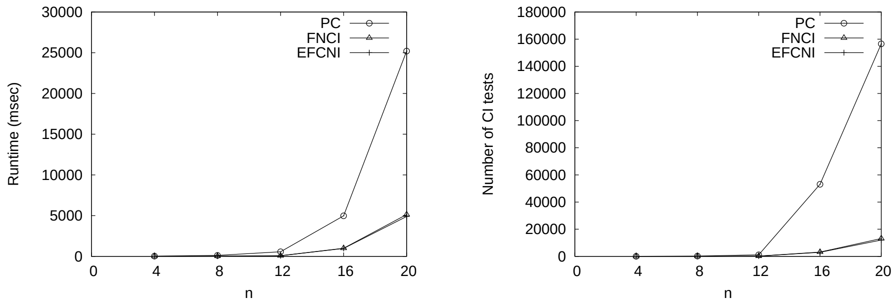
Table 6. Number of erroneous edges in a changing event stream over the six observation points t_1 through t_6 (datasets: DS_4 and DS_5).

6.3.2 Comparison of the running time of the PC, FCNI, and EFCNI algorithms.

6.3.2.1 ***When the events arrive in order***

Figure 3(a) shows the average running time of the EFCNI, FCNI, and PC algorithms for varying number of event types in the synthetic datasets. In all cases, the running time of the EFCNI algorithm is the shortest while the running time of the PC algorithm is the longest. Clearly, the temporal precedence information helps to reduce the size of condition set and the number of edges for CI tests in both the FCNI and EFCNI algorithms. In addition, the EFCNI algorithm sorts the edges based on their temporal strength and then tests the conditional independence of the weaker edges, which are more likely to fail the tests, earlier and therefore further reduces the running time. As the number (n)

of event types increases, the running time increases in all three algorithms, but the rate of increase is the highest for the PC algorithm and the lowest for the EFCNI algorithm. The same observation is made in the real dataset where the running time of the PC, FCNI, and EFCNI algorithms are 817, 118, and 112 msecs, respectively. These results verify the important role of temporal precedence relationships in reducing the running time.



(a) Running time for varying number of event types.

(b) Number of CI tests for varying number of event types.

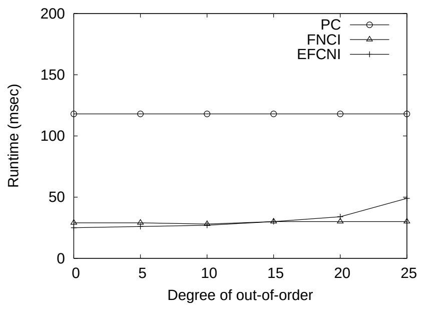
Fig. 3. Comparison of the running time of the PC, FCNI and EFCNI algorithms for in-order event streams.

6.3.2.2 ***When the events arrive out of order***

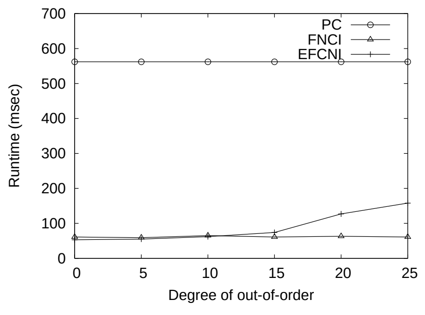
Figure 4 shows that the EFCNI algorithm performs the fastest causal network inference among the three algorithms when the event stream is in order (i.e., degree of out-of-order $d_{oo} = 0$). As d_{oo} increases, the running time of the EFCNI algorithm increases rapidly. It is due to the strategy that renders the edges with temporal strength lower than θ in the temporal network undirected. Consequently, the number of CI tests increases, resulting in an increase in the running time. On the other hand, as seen in the figure, the running time of FCNI algorithm remains short for the out-of-order event arrivals. However, the FCNI algorithm compromises the accuracy in such an event stream as discussed in Section 6.3.1. In addition, the result shows that the running time of the PC algorithm is constant as it is not affected by the out-of-order event arrivals.

6.3.2.3 ***When the event stream has changing temporal precedence statistic***

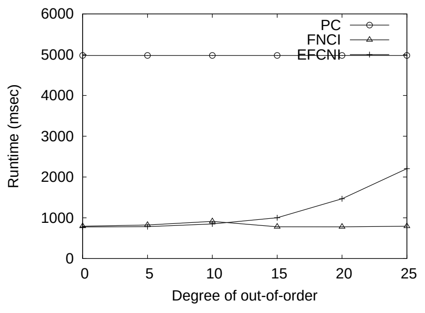
Figure 5 shows that the EFCNI algorithm performs the fastest causal network inference among the three algorithms over the event stream with changing temporal precedence statistic. In the figure, for all values of n, the FCNI and PC algorithms perform the CI tests for the causal network inference every time a new batch of events arrives. On the other hand, the EFCNI algorithm performs it only when the precedence statistic changes significantly enough in the event



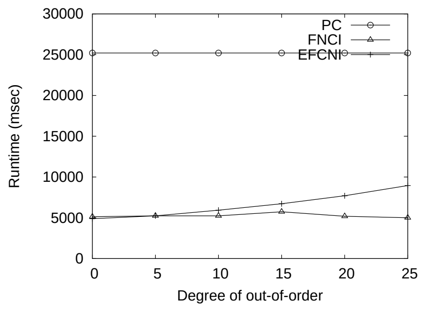
(a) n = 8.



(b) n = 12.



(c) n = 16.



(d) n = 20.

Fig. 4. Comparison of the running time of the PC, FCNI, and EFCNI algorithms for out-of-order event streams.

stream. (C_p is greater than δ at t_1, t_2, t_5, and t_6.) The EFCNI algorithm skips the causal inference at t_3 and t_4, which helps to reduce the overall running time.

6.3.3 Comparison of the number of CI tests of the PC, FCNI, and EFCNI algorithms.

6.3.3.1 ***When the events arrive in order***

Figure 3(b) shows that the EFCNI algorithm performs fewer CI tests than the PC and FCNI algorithms in all synthetic datasets. The CI tests are decreased by reducing the size of the condition set and the number of edges to test with the help of the temporal precedence information. In addition, the sorting of the edges (based on the their temporal strengths) helps to reduce the number of CI tests. With an increase in the number of event types (n), the rate of increase in the number of CI tests of the PC algorithm is much higher than that of the EFCNI and FCNI algorithms. A similar observation is made in the real dataset where the number of CI tests of the PC, FCNI, and EFCNI algorithms are 1239, 192, and 176, respectively. These results confirm the important role of temporal precedence relationships in reducing the number of CI tests. Note the result of CI tests (Figure 3(b)) looks almost the same as that of the running

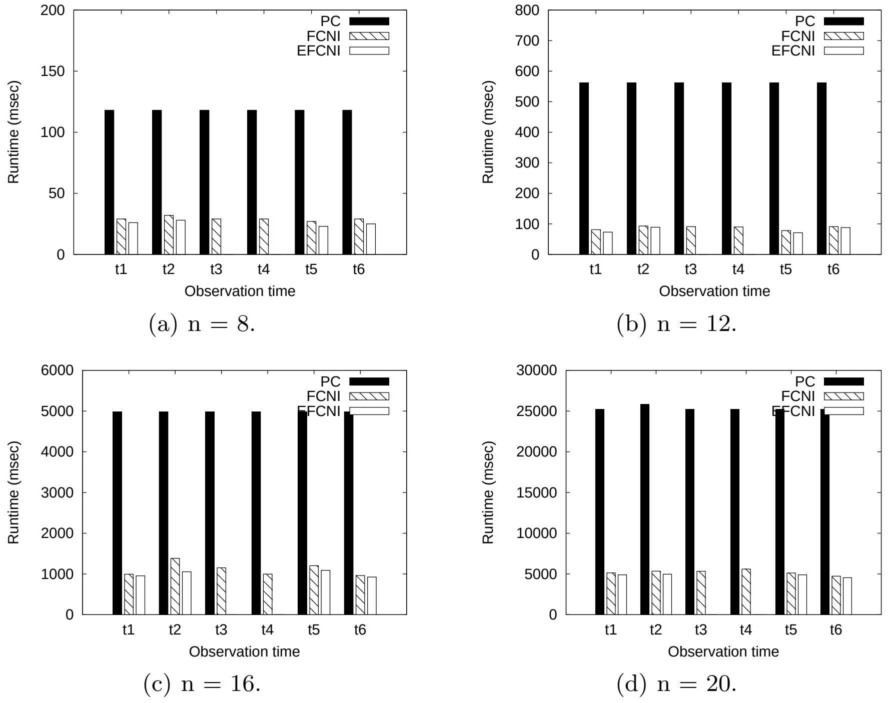


Fig. 5. Comparison of the running time of the PC, FCNI, and EFCNI algorithms for changing event streams.

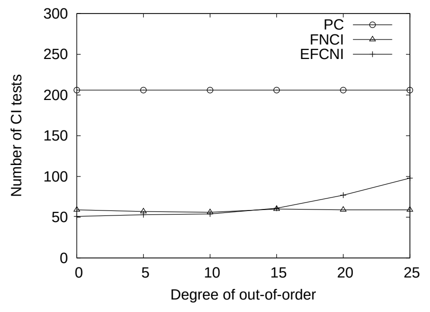
time (Figure 3(a)). This demonstrates that CI tests are the major performance bottleneck and validates the key idea of our work that reducing the number of CI tests reduces the run time.

6.3.3.2 ***When the events arrive out of order***

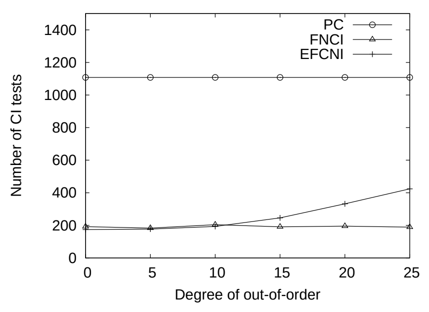
Figure 6 shows that, as the degree of out-of-order increases, the number of CI tests of the EFCNI algorithm increases. A higher degree of out-of-order leads to the temporal strengths of more edges lower than the temporal confidence threshold (i.e., θ), and this results in rendering more edges undirected and therefore performing more CI tests. Consequently, as the degree of out-of-order increases, the number of CI tests of the EFCNI algorithm becomes closer to that of the PC algorithm. Note that the PC algorithm is not affected by the out-of-order event arrivals and, thus, the number of CI tests does not change for varying degree of out-of-order. The results also show that the FCNI algorithm performs the smallest number of CI tests when the events arrive out of order. However, its accuracy is the worst among the three algorithms as discussed in Section 6.3.1.

6.3.3.3 ***When the event stream has changing temporal precedence statistic***

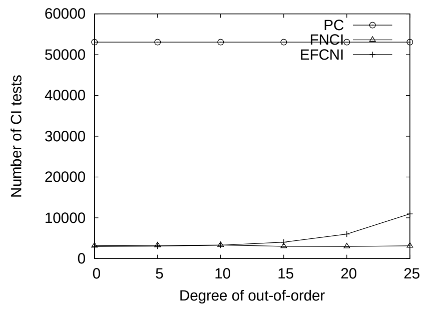
Figure 7 shows that the EFCNI algorithm performs the smallest number of CI



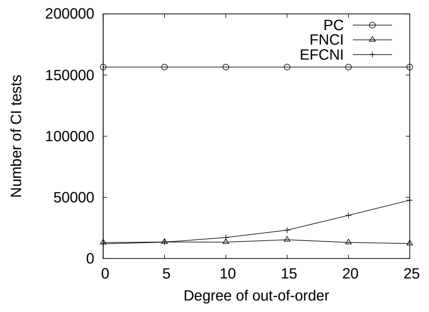
(a) n = 8.



(b) n = 12.



(c) n = 16.



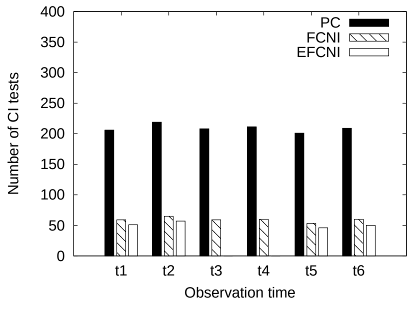
(d) n = 20.

Fig. 6. Comparison of the number of CI tests of the PC, FCNI, and EFCNI algorithms for out-of-order event streams.

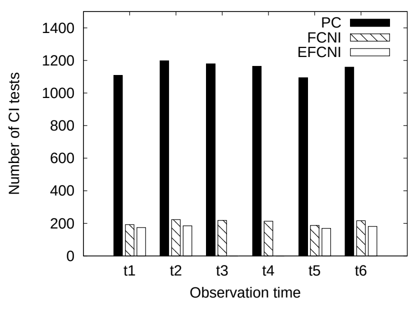
tests among the three algorithms over the event stream with changing temporal precedence statistic. As discussed earlier, the FCNI and PC algorithms regenerate the causal network with the arrival of every new batch of events while the EFCNI algorithm does it only when the change in the event stream (i.e., C_p) is greater than δ (as seen at t_1, t_2, t_5, and t_6). As a result, the EFCNI algorithm skips causal inference computations involving a large number of CI tests at t_3 and t_4 where the value of C_p is less than δ. As expected, the number of CI tests is highest for the PC algorithm at every observation point t_1 through t_6 due to its highest computational complexity.

6.3.4 Summary of experiment results

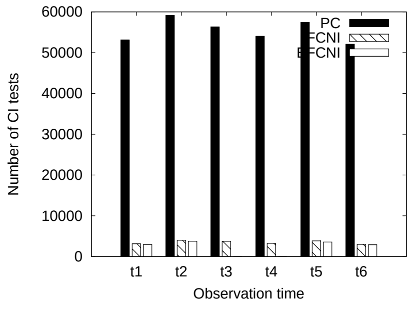
The EFCNI algorithm is faster than the FCNI algorithm for an event stream where the events arrive in order. The EFCNI algorithm enhances the FCNI algorithm with two additional strategies to reduce the number of CI tests. It has been demonstrated that the CI tests are the performance bottleneck and thus the reduction in the number of CI tests is the key to decreasing the running time of the algorithm. Moreover, unlike the FCNI algorithm which requires events to arrive in order, the EFCNI algorithm can perform the causal network inference accurately even when the events arrive out of order. **As the degree of out-of-**



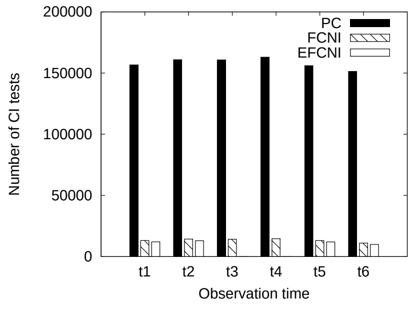
(a) n = 8.



(b) n = 12.



(c) n = 16.



(d) n = 20.

Fig. 7. Comparison of the number of CI tests of the PC, FCNI, and EFCNI algorithms for changing event streams.

orderness d_{oo} increases, the accuracy of the EFCNI algorithm comes at the expense of the running time (i.e., the number of CI tests) to some extent.

The EFCNI algorithm is much faster than the PC algorithm in all experiments. **In some scenarios (e.g., medical applications like patient health tracking), the accuracy of the result may be more important than the runtime. In case the accuracy of the EFCNI algorithm is not satisfactory in such scenarios – for example, if the event stream has many out-of-order events (i.e., with large d_{oo}) – then the OATNI algorithm can be tweaked to increase the EFCNI algorithm's accuracy by setting θ to a larger value (e.g., toward 100%). A larger θ value forces the edges more to be undirected and, therefore, makes the EFCNI algorithm behave more like the PC algorithm, thus achieving higher accuracy.**

Furthermore, the EFCNI algorithm saves time by avoiding causal inference computations when there is not significant enough changes in the statistic of the event stream.

7 Conclusion and Future Work

In this paper, we presented a novel strategy to exploit temporal precedence relationships to learn the causal network over event streams. First, we introduced the *Order-Aware Temporal Network Inference* algorithm to model temporal precedence information. Then, we presented the *Enhanced Fast Causal Network Inference* algorithm to reduce the running time of learning causal network by reducing the number of performed CI tests significantly. These algorithms efficiently handle the event streams even if the events are out of order and saves the running time further by performing causal inference only if the temporal precedence statistic changes significantly enough. We showed the experiment results to validate our approach by comparing against the state-of-the-art PC and FCNI algorithms.

There are a number of future work in the plan. First, we plan to support cyclic causality. Second, we plan to investigate the effect of concept drift in the causal network inference so that the computations are performed only among the event types affected by the changes. **Third, we plan to perform the experiments on datasets with a much larger number (i.e., hundreds to thousands) of event types to show the practicality of our proposed algorithms in a big data environment.**

References

1. Barga, R.S., Goldstein, J., Ali, M.H., Hong, M.: Consistent streaming through time: A vision for event stream processing. In: Proceedings of the Third Biennial Conference on Innovative Data Systems Research. CIDR'07 (2007) 363–374
2. Zhao, Y., Strom, R.: Exploitng event stream interpretation in publish-subscribe systems. In: Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing. PODC '01 (2001) 219–228
3. Johnson, T., Muthukrishnan, S., Rozenbaum, I.: Monitoring regular expressions on out-of-order streams. In: Proceedings of the IEEE 23rd International Conference on Data Engineering. ICDE'07 (2007) 1315–1319
4. Li, M., Liu, M., Ding, L., Rundensteiner, E.A., Mani, M.: Event stream processing with out-of-order data arrival. In: Proceedings of the 27th International Conference on Distributed Computing Systems Workshops. ICDCSW '07, Washington, DC, USA, IEEE Computer Society (2007) 67–74
5. Liu, M., Li, M., Golovnya, D., Rundensteiner, E., Claypool, K.: Sequence pattern query processing over out-of-order event streams. In: Proceedings of the IEEE 25th International Conference on Data Engineering. ICDE '09 (2009) 784–795
6. Li, J., Tufte, K., Shkapenyuk, V., Papadimos, V., Johnson, T., Maier, D.: Out-of-order processing: A new architecture for high-performance stream systems. Proceegins of the VLDB Endowment **1**(1) (August 2008) 274–288
7. Wang, K., Yu, Y.: A query-matching mechanism over out-of-order event stream in iot. Int. J. Ad Hoc Ubiquitous Comput. **13**(3/4) (July 2013) 197–208
8. Heckerman, D.: A Bayesian approach to learning causal networks. In: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. UAI'95 (1995) 285–295

9. Ellis, B., Wong, W.H.: Learning causal Bayesian network structures from experimental data. J. American Statistics Association **103**(482) (2008) 778–789
10. Li, G., Leong, T.Y.: Active learning for causal Bayesian network structure with non-symmetrical entropy. In: Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining. PAKDD '09 (2009) 290–301
11. Meganck, S., Leray, P., Manderick, B.: Learning causal Bayesian networks from observations and experiments: A decision theoretic approach. In: Modeling Decisions for Artificial Intelligence. Volume 3885 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2006) 58–69
12. Pearl, J.: Causality: Models, Reasoning and Inference. 2nd edn. Cambridge University Press (2009)
13. Spirtes, P., Glymour, C.N., Scheines, R.: Causality from probability. In: Proceedings of the Conference on Advanced Computing for the Social Sciences. ACSS '90 (1990)
14. Spirtes, P., Glymour, C., Scheines, R.: Causation, Prediction, and Search. MIT Press (2000)
15. Cheng, J., Greiner, R., Kelly, J., Bell, D., Liu, W.: Learning Bayesian networks from data: an information-theory based approach. Artificial Intelligence **137**(1-2) (2002) 43–90
16. Pearl, J.: Causal diagrams for empirical research. Biometrika **82** (1995) 669–710
17. Spirtes, P., Meek, C.: Learning Bayesian networks with discrete variables from data. In: Proceedings of the First International Conference on Knowledge Discovery and Data Mining. KDD'95 (1995) 294–299
18. Chow, Y.S., Teicher, H.: Probability theory : independence, interchangeability, martingales. Springer-Verlag, New York (1978)
19. Prakasa Rao, B.: Conditional independence, conditional mixing and conditional association. Annals of the Institute of Statistical Mathematics **61**(2) (2009) 441–460
20. Popper, K.: The Logic of Scientific Discovery. Reprint edn. Routledge (Oct 1992)
21. Hamilton, H.J., Karimi, K.: The TIMERS II algorithm for the discovery of causality. In: Proceedings of the 9th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining. PAKDD'05, Berlin, Heidelberg, Springer-Verlag (2005) 744–750
22. Utrera, A.C., Olmedo, M.G., Callejon, S.M.: A score based ranking of the edges for the pc algorithm. In: Proceedings of the 4th European workshop on probabilistic graphical models. PGM'08 (2008) 41 – 48
23. Acharya, S., Lee, B.: Fast causal network inference over event streams. In: Data Warehousing and Knowledge Discovery. Volume 8057 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2013) 222–235
24. Chickering, D.M.: Learning equivalence classes of Bayesian-network structures. J. Machine Learning Research **2** (2002) 445–498
25. Borchani, H., Chaouachi, M., Ben Amor, N.: Learning causal Bayesian networks from incomplete observational data and interventions. In: Proceedings of the 9th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty. ECSQARU '07, Berlin, Heidelberg, Springer-Verlag (2007) 17–29
26. de Campos, L.M.: A scoring function for learning Bayesian networks based on mutual information and conditional independence tests. J. Machine Learning Research **7** (2006) 2149–2187
27. Bishop, Y.M., Fienberg, S.E., Holland, P.W.: Discrete Multivariate Analysis: Theory and Practice. MIT Press (1975)

28. Kullback, S.: Information Theory and Statistics. 2nd edn. Dover Publication (1968)
29. Verma, T., Pearl, J.: Causal networks: Semantics and expressiveness. In: Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence. UAI '88 (1988) 69–78
30. Geiger, D., Pearl, J.: On the logic of causal models. In: Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence. UAI '88 (1988) 3–14
31. Pearl, J.: Graphs, causality, and structural equation models. Sociological Methods and Research **27** (1998) 226–84
32. Tsamardinos, I., Brown, L.E., Aliferis, C.F.: The max-min hill-climbing Bayesian network structure learning algorithm. Mach. Learn. **65**(1) (October 2006) 31–78
33. Frank, A., Asuncion, A.: UCI machine learning repository. `http://archive.ics.uci.edu/ml/datasets/Diabetes` (2010)