Refining Causality: Who Copied From Whom?

Tristan Snowsill
Intelligent Systems Laboratory
University of Bristol

Nick Fyson
Bristol Centre for Complexity Sciences and ISL
University of Bristol

Tijl De Bie
Intelligent Systems Laboratory
University of Bristol

Nello Cristianini
Intelligent Systems Laboratory
University of Bristol

ABSTRACT

Inferring causal networks behind observed data is an active area of research with wide applicability to areas such as epidemiology, microbiology and social science. In particular recent research has focused on identifying how information propagates through the Internet. This research has so far only used temporal features of observations, and while reasonable results have been achieved, there is often further information which can be used.

In this paper we show that additional features of the observed data can be used very effectively to improve an existing method. Our particular example is one of inferring an underlying network for how text is reused in the Internet, although the general approach is applicable to other inference methods and information sources.

We develop a method to identify how a piece of text evolves as it moves through an underlying network and how substring information can be used to narrow down where in the evolutionary process a particular observation at a node lies. Hence we narrow down the number of ways the node could have acquired the infection. Text reuse is detected using a suffix tree which is also used to identify the substring relations between chunks of reused text. We then use a modification of the NetCover method to infer the underlying network.

Experimental results – on both synthetic and real life data – show that using more information than just timing leads to greater accuracy in the inferred networks.

Categories and Subject Descriptors: H.2.8 Database applications: data mining

General terms: Theory

Keywords: network inference, reconstruction, text reuse, suffix tree

1. INTRODUCTION

In recent years the study of complex networks and their behaviours has become recognised as having great potential. Often the behaviour of the network is a result of the behaviours of the individual nodes under the influence of the behaviour of neighbouring nodes. The spreading of infections across a network has attracted substantial interest in the areas of outbreak detection, epidemic modelling and viral marketing.

In many cases the underlying network of relations between nodes is unknown and observation of the network is either impossible, impractical or not cost-effective, *e.g.*, epidemiological networks.

In some important cases, however, it is possible to make observations of the nodes and *infer* the underlying network. Our principal example of this is in the network of information diffusion on the Internet. When a piece of information reaches an outlet, it then behaves in a way to indicate the receipt of that information. The behaviour could be something simple, such as posting a URL, as considered in [3], or it could be something complex, such as using text which is significant in some way, *e.g.*, reproducing a speech. In this paper we describe how we can identify *text reuse* (when a sizeable chunk of text already seen at one node is reproduced by another) in a large collection of documents produced by multiple outlets which are assumed to be connected by an underlying network.

Existing methods [3, 9, 8] search for and identify markers or indicators that a node has acquired some particular infection and then use only the time of infection for inference. This essentially assumes that the underlying infection does not evolve in any way, and that the indicators may only be different due to noise. This assumption is, we believe, a waste of valuable information as we can envisage cases where infections *do evolve*. Importantly the evolution of infections will not be uniform across different branches of the propagation tree which means we can discover much more about the way infections have spread by studying the states of the infection at the nodes. This concept and an example of its power are our key contributions to the field in this paper.

In our particular example of text reuse we identify that it is only possible to copy text from another node if it already produced that text, and that the "infection" will evolve as different outlets reuse certain chunks of text and extend it. Indeed, certain nodes carrying the "infection" may share very little or no text with the node originally "infected".

This is just one example of how infections can evolve. Textual infections can evolve in other ways, *e.g.*, when short quotes are copied they may be hand-copied, resulting in transmission errors; when a news outlet reports news, the presence or absence of quotes can be an excellent indicator

KDD'11, August 21–24, 2011, San Diego, California, USA.

of the source of the information. We also reiterate that such information should benefit *all* network inference methods.

Outline of this paper.

In this paper we detail an example of how we can improve the inference of networks of outlets from text reuse data by identifying and using information about how reused text evolves. This example motivates the general approach of improving network inference by using additional features. In Sec. 2.1 we establish notation which will be used throughout the paper. In Sec. 3 we describe our method for inferring networks using text reuse and in Sec. 4 we show experimentally that the use of additional features improves network inference.

2. PRELIMINARIES

In this section we introduce notation which will help us to solve the text reuse problem which is used as an example of how network inference can be improved by using additional features of the observed data (in this case, how the text that is reused evolves as it progresses through the network). We also include Sec. 2.2, which introduces suffix trees for readers unfamiliar with them.

2.1 Memes, markers and reports

In the motivating example of this paper we are attempting to infer how online news outlets influence each other by detecting text reuse. Our notation is therefore tailored towards this application. We denote the set of outlets as $\mathcal{W}$ (from websites). Each outlet has a unique identifier. We denote the set of documents or articles as $\mathcal{D}$ (from documents). Each document similarly has a unique identifier. Each document comes from a single outlet, and so a function is defined to identify the outlet which published a document, $w : \mathcal{D} \to \mathcal{W}$.

Each document is a string of symbols drawn from an alphabet, Σ, *i.e.*, $d \in \mathcal{D}$, $d = \Sigma^\star$. The length or size of a document is defined as the number of symbols in the document. No limit is imposed on the size of the alphabet or of the documents. We choose the alphabet to be the set of words used (rather than the immediately obvious choice of the set of characters used).

We suppose that there is a simple underlying infection model which causes text to be copied between outlets, as adopted in, *e.g.*, [3, 9]. We additionally suppose that there is a finite set of outlets which can infect a given outlet. This leads to a network of influence between outlets. For simplicity we assume the network does not change over time. The network is described in graph notation as $G_T = (\mathcal{W}, E_T)$, where E_T is the *true* set of edges between outlets, such that if $(w_1, w_2) \in E_T$ then it is possible for w_1 to infect w_2. We assume the set of outlets is fully known and that we only need to infer the edges between outlets to produce $G_R = (\mathcal{W}, E_R)$, the *reconstructed* network. This is a formulation common to recent papers [8, 9] attempting to solve this problem.

As we do not expect the text to always be copied in the same way (small extracts may be taken, editorial comments may be injected), we say the general concept which is copied is a *meme*, which spreads from one outlet to others. It is difficult to comprehensively define a meme in this context (as it is indeed difficult to define a meme in general), so we make some simplifications.

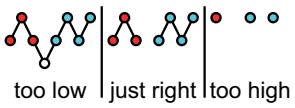

Figure 1: Setting l affects the connected-ness of the marker partial order graph, affecting what is considered a single meme

A string of n symbols drawn from Σ is termed an n-gram. We assume the reader is familiar with the substring, prefix and suffix relations.

We attempt a formal definition of a meme by stating that if one n-gram is a substring of another (and is sufficiently long that it is not a substring of numerous otherwise unrelated n-grams) then they belong to the same meme. We define *belonging to the same meme* as a transitive and symmetric relation, *e.g.*, if the n-gram x belongs to the same meme as y, and z belongs to the same meme as y, then x, y and z all belong to the same meme. A meme can be denoted as a partial order $(X, \prec)$, where X is the set of n-grams belonging to the meme and $\prec$ is the substring relation. The Hasse diagram of the partial order of a meme will be connected.

When an n-gram contained in the partial order of a meme is found in a document, we call the n-gram a *marker* because it marks that the meme has "infected" the outlet. The description of the observation is denoted a *report*.

It is necessary for us to tie down the concept of markers further. We therefore introduce two constraints which must be satisfied for an n-gram to belong to *any* meme (and hence must also be satisfied for all markers).

Constraint 1 (Minimum length). *The length of the n-gram must be at least l for it to belong to a meme. The number of n-grams satisfying the length constraint is monotonically decreasing for increasing values of l.*

Constraint 2 (Outlet support). *The n-gram must appear in documents from at least C_w outlets. The number of n-grams satisfying the outlet support constraint is monotonically decreasing for increasing values of C_w.*

There are some natural choices of constraint parameters which can be identified. One natural choice for C_w is 2, which simply ensures that there is evidence that text has been reused. The choice of l is more subjective as low values of l could result in poorly defined memes (see Fig. 1).

A further consideration is that choosing lower support constraints will increase the size of the set of markers and will therefore increase computation time.

For a given set of documents and a choice of constraint parameters there will be a corresponding set of markers. A document may contain a number of markers. The set of markers for a given set of documents is denoted $\mathcal{M} = \mathcal{M}(\mathcal{D})$. The substring partial order is defined over $\mathcal{M}$ and may be computed and represented as a directed graph. Each connected component in the undirected equivalent of this graph is a meme.

The reason why we seem so concerned about the substring relation is that it helps us to identify how a meme has evolved by using simple rules to reduce the number of possible ways an outlet could have been infected. By eliminating

potential edges from our network reconstruction we improve our inference.

2.2 Suffix trees

Suffix trees [17] are a powerful data structure used frequently in combinatorial pattern matching and computational genomics [10]. Their key strength is the ability to index in linear space all the substrings of a given text, when naïve methods would require quadratic space. They can also be constructed in linear time with a constant alphabet size [12, 16], with logarithmic time complexity in the alphabet size when the alphabet size is not constant. *Generalized suffix trees* are a superimposition of multiple suffix trees to create a single tree indexing all the substrings of a set of texts, and require linear space and time to construct in the total length of texts [16]. We give a brief description of the structure of the suffix tree as it is a vital component in the text reuse detection component of our example.

The suffix tree of a text, T, is a rooted tree where edges are labelled with non-empty substrings from the text. A node in the suffix tree represents a substring of the text which is formed by concatenating the edge labels on the path from the root to the node. Each leaf node represents a suffix of the text and each suffix of the text is represented by a unique leaf node. No two edges outgoing from a node begin with the same symbol. Edge labels are stored using a pair of indices referring to the start and end positions of a substring in the text. Every substring in the text exists either as a node in the suffix tree (an *explicit node*), or partway down an edge (an *implicit node*). Many suffix tree implementations include the concept of a *suffix link*, which maps each internal node to another node in the tree whose corresponding substring in the text is the largest proper suffix of the substring of the original node, *i.e.*, following a suffix link removes the first symbol from the corresponding substring.

Suffix trees can be preprocessed [13] in linear time (independent of the alphabet size) to allow constant time *lowest common ancestor* (LCA) queries which in the context of a suffix tree give the longest common prefix of two substrings in the text.

Due to poor memory locality, suffix trees are often thought only to be appropriate for in-memory applications. These issues can, however, be mitigated even in the case where the alphabet is large and unbounded. We have implemented Ukkonen's algorithm [16] on hard-disk successfully by separating data for nodes and edges effectively to improve memory locality and using memory mapping, which allows files on hard-disk to be accessed randomly without loading them entirely into memory [15].

3. METHOD

In this section we describe our method for inferring networks of influence using text reuse. First we give an outline of our method, then we give more detailed descriptions of components of our method, including complexity analysis.

3.1 Outline

Our method attempts to infer the network of influence between text sources by identifying text reuse between the sources. We divide our method into three components: *data collection*, *text reuse detection*, and *network reconstruction*.

The data collection component handles gathering the data from the text sources and the processing necessary to turn text cluttered with markup, advertisements and site layout code into plain text. The final output of this component is plain text documents; *i.e.*, the document set, $\mathcal{D}$.

Once the document set has been collated it is passed to the component for text reuse detection. This component uses a suffix tree data structure to identify potential cases of text reuse. There is then further processing to eliminate redundant candidates until we are left with the marker set, $\mathcal{M}$. While the substring relation is defined on $\mathcal{M}$ it is not computed when $\mathcal{M}$ is found, so this must also be calculated, to produce $(\mathcal{M}, \prec)$, which is represented by the boolean matrix M, in which $m_{i,j} = true$ iff marker i is a proper superstring of marker j (note here that we are representing the superstring ordering; the substring ordering would be represented by M^{T}, the transpose of M). Finally the *coverage* of $\mathcal{M}$ in $\mathcal{D}$ is computed in the form of the boolean matrix C, in which $c_{i,j} = true$ iff marker i appears in document j.

The network reconstruction component can be further split into two subcomponents: *candidate generation* and *reconstruction method*. The candidate generation component takes M and C and produces a partially ordered set of reports, $\mathcal{R}$. The partial ordering on this set encodes whether a report can be "explained" by an earlier report and hence produces a set of candidate edges between reports for the reconstruction method. These candidate edges are specified in such a way as to indicate which edges in the outlet network could explain the presence of a marker in a document. The reconstruction method attempts to select edges for the outlet network to explain as many of the markers observed as possible. The reconstruction method is a modification of that described in [8].

3.2 Data collection

The data that needs to be collected for our method is a set of documents from multiple sources. For each document the publication time and the source must be known, as well as the full plain text of the article. There are several contexts in which this data could be found, *e.g.*, news feeds, social media, blogs, emails. In this paper we focus on the example of news feeds. Leskovec et al. [11, 9] have investigated combining news feeds and blogs by using the Spinn3r API [1], a commercial data collection service.

As part of a larger project investigating multilingual news media around the world and in Europe in particular, our research group has produced a system which regularly visits over 1 100 outlets and a total of over 3 500 RSS/Atom feeds to collect over 47 500 articles per day. Of these over 7 000 articles per day are written in English, the remainder being written in a number of European languages.

We are making four weeks of news articles written in English available to invite others to test our method independently [2]. To avoid any potential copyright infringement, however, the data is encoded using a word-to-integer mapping, which is used in our method and should not negatively impact on any attempts to apply other methods to the data.

3.3 Text reuse detection

To detect text reuse in the document set we use a *generalized suffix tree*, a data structure described in Sec. 2.2. We insert all the documents in the document set into the suffix tree. We then identify all n-grams which satisfy the constraints l and C_w in the following way:

Firstly we compute the *word depths* of all the nodes in

the suffix tree. This is done simply in linear time in the number of nodes in the suffix tree – which we will denote as n within this paper – by using a depth first search. Our algorithm uses *parent links* which are calculated by most suffix tree construction algorithms. It also requires constant time look-up for the number of children of a node, which again is available with most suffix tree implementations or can be precomputed in linear time.

We then iterate in a bottom-up fashion, starting at the leaf nodes and propagating the document and outlet IDs upwards through the tree following parent links. The outlet support of a node is monotonically increasing on the path from that node to the root. The word depth of a node is monotonically decreasing on the path from that node to the root. In this way the outlet support constraint filters out nodes from the bottom of the tree and the length constraint filters out nodes from the top of the tree.

For each node satisfying all constraints we wish to store the sets of documents and outlets in which the string corresponding to the node is found. We do not need to store these sets for nodes not meeting the constraints. Alg. 1 calculates the set of nodes corresponding to n-grams which satisfy the constraints and also calculates the *coverage* of each node (the set of documents containing the n-gram).

```
Data: T, the suffix tree of D.
Data: wordDepth, the word depths of nodes in T.
Data: l, C_w
Result: coverage
toProcess ← {};
foreach leaf ∈ T.LeafNodes() do
    parent ← T.ParentOf(leaf);
    if wordDepth(parent) < l then
        next leaf;
    toProcess ← toProcess ∪ {parent};
    increment nbChildrenProcessed(parent);
    coverage(parent) ← coverage(parent) ∪ coverage(leaf)
    delete coverage(leaf);
while toProcess ≠ ∅ do
    foreach node ∈ toProcess do
        if nbChildrenProcessed = T.NbChildren(node)
        then
            parent ← T.ParentOf(node);
            if wordDepth(parent) ≥ l then
                toProcess ← toProcess ∪ {parent};
                increment nbChildrenProcessed(parent);
                coverage(parent) ←
                coverage(parent) ∪ coverage(node)
            toProcess ← toProcess \ {node};
            if not Satisfies( coverage(node) ) then
                delete coverage(node);
```

Algorithm 1: Identify n-grams satisfying constraints

Although Alg. 1 identifies all n-grams satisfying the constraints, it also produces a set of n-grams which has some redundancy in the sense that there are likely to be n-grams which can be extended without altering the coverage. The fact that these n-grams are reused is explained entirely by a longer n-gram being used, so we filter them out. This filtering step involves simply tracing parent links and suffix links and removing redundant nodes. Tracing along parent links for redundancy removes 10–30% of candidate n-grams. Tracing along suffix links for redundancy removes a further 95–99% of candidate n-grams.

The final step in the text reuse detection component is to compute and store the substring partial order over the marker set, $\mathcal{M}$. We use the fact that if the lowest common ancestor (LCA) of two nodes in a tree is one of those nodes then that node is an ancestor of the other, and in the context of the suffix tree it corresponds to a prefix of the other node. If x is a substring of y then it must be a prefix of a suffix of y. We therefore use constant time LCA queries on all the suffixes of y which are at least as long as x to determine whether x is a substring of y and repeat this for all pairs of markers. To assist in memory locality it is also useful to use the transitive property of the substring relation to partially complete the substring partial order over $\mathcal{M}$.

3.4 Network reconstruction

The network reconstruction task is divided into *candidate generation* and the *reconstruction method*.

3.4.1 Candidate generation

Once we have identified the set of markers, $\mathcal{M}$, and the coverage of those markers, we then need to generate candidate edges for the network of outlets. We only want to provide a candidate edge from one outlet to another if there is evidence of text reuse between them and that it is possible for one outlet to have copied from the other.

As described in Sec. 2.1 we use a concept of *reports* to describe when a marker is observed in a document. A report is then a tuple of the marker and the document. The set of reports $\mathcal{R}$ is a subset of the Cartesian product of $\mathcal{M}$ and $\mathcal{D}$, *i.e.*, $\mathcal{R} \subseteq \mathcal{M} \times \mathcal{D}$. The coverage matrix gives us an obvious set of reports, $\mathcal{R}_{\mathrm{C}} = \{(m, d) \mid c_{m,d} = true\}$. As well as a partial order over markers, there is a total order over documents, where $d_i > d_j$ means d_i was published *before* d_j. We can define a partial order over a given set of reports, in particular we define

$$m_1 \succeq m_2,\, d_1 \geq d_2 \;\Leftrightarrow\; r_1 = (m_1, d_1) \succeq r_2 = (m_2, d_2)\,,$$
$$m_1 = m_2,\, d_1 = d_2 \;\Leftrightarrow\; r_1 = r_2.$$

It is assumed that if $d_i \not> d_j$ and $d_i \not< d_j$ then d_i and d_j are the same document (*i.e.*, each individual document has a unique position in the total ordering).

$\mathcal{R}$ will be used by the reconstruction method as a set of candidate edges by considering the directed graph representation of the partial order. The reconstruction method will only be able to place an edge from one outlet to another if there is at least one pair of reports which supports this.

We would like to make $\mathcal{R}$ as small as possible while still generating all candidate edges which the data suggests. The reason for this is that the reconstruction method assigns equal value to explaining each report and hence redundant reports may skew the decisions of the reconstruction method. We therefore pose a minimization problem where we attempt to make $\mathcal{R}$ optimally small while respecting the constraints which are encoded in $\mathcal{R}_{\mathrm{C}}$.

In this section we show that the minimization problem can be tackled by a greedy approach and present both an algorithm to find the optimal solution (Alg. 2) and a more efficient heuristic (Alg. 3) which performs well in practice. Readers may safely skip to Sec. 3.4.2.

Formally, our aim is to find $\mathcal{R}$ satisfying the following minimization problem:

$$\min_{\mathcal{R}\subseteq\mathcal{R}_{\mathrm{C}}} |\mathcal{R}| \tag{1}$$

subject to two constraints. The first constraint ensures that if a marker is observed in a document then that fact is recorded in a report. The report may have a marker which is a superstring of the report originally considered. Formally:

$$\forall r = (m,d) \in \mathcal{R}_{\mathrm{C}}, \exists r' = \left(m',d\right) \in \mathcal{R} \,|\, m' \succeq m. \tag{2}$$

The second constraint ensures that if an edge between outlets could be inferred from $\mathcal{R}_{\mathrm{C}}$ then it will also be inferrable from $\mathcal{R}$. Formally:

$$\begin{aligned}(\exists r_1 = (m_1,d_1), r_2 = (m_2,d_2) \in \mathcal{R}_{\mathrm{C}} \,|\, r_1 \succ r_2) \rightarrow \\ \exists r_1' = \left(m_1',d_1\right), r_2' = \left(m_2',d_2\right) \in \mathcal{R} \,|\, r_1' \succeq r_1, r_1' \succ r_2' \succeq r_2.\end{aligned} \tag{3}$$

We say that a report, $r = (m,d)$ is a *maximal* report if

$$\nexists r' = \left(m',d\right) \in \mathcal{R}_{\mathrm{C}} \;|\; m' \succ m.$$

We say that a report can be safely eliminated from $\mathcal{R}$ if both Eqs. 2 and 3 hold before and after the elimination. We let $\mathcal{R}_k$ denote the set of reports after k reports have been safely eliminated, such that $\mathcal{R}_0 = \mathcal{R}_{\mathrm{C}}$.

We now develop some theories to help us construct an optimal algorithm.

THEOREM 1. *Eq. 2 cannot be satisfied if maximal reports are removed.*

PROOF. A maximal report, $r = (m,d)$, is one such that there is no $r' = (m',d)$ with $m' \succ m$. So if r is removed $\nexists r'$ with $m' \succeq m$. □

THEOREM 2. *Provided edges are eliminated safely, we can replace $\mathcal{R}_{\mathrm{C}}$ in Eqs. 2 and 3 with $\mathcal{R}_k$.*

PROOF. By induction. □

This means we only need to examine report pairs in Eq. 3 which remain in the current set of reports.

We now denote the set of reports which can individually be safely eliminated from $\mathcal{R}_k$ as $\mathcal{R}_k^-$, $\mathcal{R}_k^- \subset \mathcal{R}_k$.

THEOREM 3. *If $r \in \mathcal{R}_i^-$ is removed from $\mathcal{R}_i$, producing $\mathcal{R}_{i+1}$ then $\mathcal{R}_{i+1}^- = \mathcal{R}_i^- \setminus \{r\}$.*

PROOF. Clearly $r \notin \mathcal{R}_{i+1}^-$ as $r \notin \mathcal{R}_{i+1}$. Now we must show that

$$\nexists r_p \neq r \;|\; r_p = (m_p,d_p) \in \mathcal{R}_i^-, r_p \notin \mathcal{R}_{i+1}^-,$$

i.e., that r is the *only* report appearing in $\mathcal{R}_i^-$ not appearing in $\mathcal{R}_{i+1}^-$. There must be a $r_q = (m_q,d_q) \in \mathcal{R}_i$, $r_q \succ r_p$, such that

$$\exists \left(r_q', r_p'\right) \in \mathcal{R}_i^2,\, r_q' = \left(m_q',d_q\right),\, r_p' = \left(m_p',d_p\right),$$

$$r_q' \succeq r_q,\, r_p' \succ r_p,\, r_q' \succ r_p',$$

but that $\left(r_q', r_p'\right) \notin \mathcal{R}_{i+1}^2$. This means that either $r_q' = r$ or $r_p' = r$. If $r_q' = r$ then r_q' is not a maximal report and there will be a $r_q'' \succ r_q' \succeq r_q$ and hence r_p can be safely eliminated, *i.e.*, $r_p \in \mathcal{R}_{i+1}^-$. If $r_p' = r$ then as r was in $\mathcal{R}_i^-$,

$$\exists \left(r_q'', r_p''\right) \in \mathcal{R}_{i+1}^2,\, r_q'' \succeq r_q' \succeq r_q,\, r_p'' \succ r_p' \succ r_p,\, r_q'' \succeq r_m'',$$

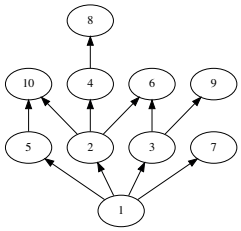

Figure 2: An example of a Hasse diagram of a partially ordered set (in this case the natural numbers up to 10 ordered by divisibility)

therefore $\left(r_q'', r_p''\right)$ satisfies Eq. 3, *i.e.*, $r_p \in \mathcal{R}_{i+1}^-$. For completeness one must also show that $\nexists r_p \in \mathcal{R}_{i+1}^- \mid r_p \notin \mathcal{R}_i^-$, *i.e.*, that removing a report does not increase the set of reports that can be safely eliminated; this can be shown as a result from Theorem 2. □

This means we only need to examine each report once to see whether it can be safely eliminated and that we obtain Corollary 1.

COROLLARY 1. *For a given $\mathcal{R}_{\mathrm{C}}$ there is a unique minimal set $\mathcal{R}^\star \subseteq \mathcal{R}_{\mathrm{C}}$ satisfying Eqs. 2 and 3.*

We briefly describe the Maximal Comparable Elements (MCE) and Lowest Greater Elements (LGE) algorithms. MCE finds the subset of elements in a partially ordered set which are comparable to a given element and maximal in the partial order. LGE finds the subset of elements in a partial order which are comparable to and strictly greater than a given element and as low in the partial order as possible. In Fig. 2, $\mathrm{MCE}(2) = \{6,8,10\}$ and $\mathrm{LGE}(2) = \{4,6,10\}$. It is simple to devise algorithms to solve these problems, and particularly efficient algorithms are possible when the transitive reduction and transitive closure of the directed graph representing the partial order are known.

Given the theorems proved in this section it is possible to use a local examination method where each report is assessed for suitability for removal independently of the removal status of other reports. Pseudocode for such a method is presented in Alg. 2.

While Alg. 2 solves the optimization problem, it is in practice a little slow and it is possible to derive a faster heuristic.

We denote a report, $r = (m,d)$, as a *source report* if

$$\nexists \left(m,d'\right) \in \mathcal{R}_{\mathrm{C}} \,|\, d' > d,$$

i.e., if d is the earliest document containing m. We additionally denote a report as a *maximal source report* if it is a source report and a maximal report. If a report is not a source report then it is denoted a *carrier report*.

We can use a simple heuristic to reduce $|\mathcal{R}|$ without violating Eqs. 2 and 3, as described in Algorithm 3. This heuristic observes that non-maximal source reports can be eliminated from $\mathcal{R}_{\mathrm{C}}$ without violating the contraints on $\mathcal{R}$. In a large experiment we found the heuristic ran 162 times faster than Alg. 2.

3.4.2 Reconstruction method

We now describe the way in which the partially ordered set of reports, $\mathcal{R}$, is used to infer the network of outlets. As

Data: $\mathcal{M}$, $\mathcal{D}$, $\mathcal{R}_{\mathrm{C}}$
Result: $\mathcal{R}^{\star} \subseteq \mathcal{R}_{\mathrm{C}}$, $\mathcal{R}^{\star} = \arg\min_{\mathcal{R} \subseteq \mathcal{R}_{\mathrm{C}}} |\mathcal{R}|$
$k \leftarrow 0$;
$\mathcal{R}^{(k)} \leftarrow \mathcal{R}_{\mathrm{C}}$;
foreach $r_2 = (m_2, d_2) \in \mathcal{R}_{\mathrm{C}}$ **do**
  $\mathcal{M}_{d_2} \leftarrow \left\{ m \mid (m, d_2) \in \mathcal{R}^{(k)} \right\}$;
  $M_2' \leftarrow \mathrm{LGE}_{\mathcal{M}_{d_2}}(m_2)$;
  if $M_2' = \emptyset$ **then**
    next r_2;
  foreach $r_1 = (m_1, d_1) \in \mathcal{R}_{\mathrm{C}} \mid r_1 \succ r_2$ **do**
    thisSatisfied $\leftarrow false$;
    foreach $m_2' \in M_2'$ **do**
      $r_2' \leftarrow (m_2', d_2)$;
      $\mathcal{M}_{d_1} \leftarrow \left\{ m \mid (m, d_1) \in \mathcal{R}^{(k)} \right\}$;
      $M_1' \leftarrow \mathrm{MCE}_{\mathcal{M}_{d_1}}(m_1)$;
      foreach $m_1' \in M_1'$ **do**
        $r_1' \leftarrow (m_1', d_1)$;
        if $r_1' \succeq r_2'$ **then**
          thisSatisfied $\leftarrow true$;
          next r_1;
    if not thisSatisfied **then**
      next r_2;
  /* It is safe to eliminate r_2 */
  $\mathcal{R}^{(k+1)} \leftarrow \mathcal{R}^{(k)} \setminus \{r_2\}$;
  $k \leftarrow k + 1$;

Algorithm 2: Algorithm to minimize $|\mathcal{R}|$

this is a development of the method in [8] we refer the reader to that paper for details and proofs.

As mentioned previously the partially ordered set of reports can be viewed as a directed graph between reports. The partial order can be found using all-pairs comparison. This directed graph will be closed under transitivity as the partial order itself is transitive. As in [8] we formulate the task of inferring the network as a set covering task. We have a universe set, $\mathcal{A} =$

$$\left\{ r = (m, d) \in \mathcal{R} \mid \exists r' = \left(m', d'\right) \in \mathcal{R}, m' \succeq m, d' > d \right\},$$

i.e., the set of reports which can be explained by reports from other documents. We also have a family, $\mathcal{B}$, of subsets of $\mathcal{R}$, each of which corresponds to the set of reports which will be explained if an edge is inferred between two outlets,

$$\mathcal{B} = \{B_{u,v} \mid u, v \in \mathcal{W}\}$$
$$B_{u,v} = \{r = (m, d) \in \mathcal{R} \mid \exists r' = \left(m', d'\right), r' \succeq r, w\left(d'\right) = u, w\left(d\right) = v\}.$$

Our goal is to find the smallest subfamily, $\mathcal{C} \subseteq \mathcal{B}$ such that $\bigcup \mathcal{C} = \mathcal{A}$. This formulation is very similar to the formulation in [8], the principal difference being that edges must not only respect the time directionality of the relations between outlets but also the textual evolutionary directionality of those relations. This means that there are fewer candidate edges from which the algorithm can choose when trying to explain reports. The set covering problem is known to be NP-hard but is well approximated using a greedy approach [5, 14].

The reconstruction method calculates a score for each potential edge which can be added to the outlet network equal to the number of additional reports that would be explained by adding that edge to the network. These scores decrease monotonically as edges are added to the network (as adding edges cannot un-explain reports) and hence a priority queue is an ideal structure to store these scores. While there are still unexplained reports the algorithm grabs the edge with the highest score from the priority queue and adds that to the network. It then recalculates scores in a lazy manner by grabbing an edge, recalculating its score then updating its score in the priority queue. If the algorithm grabs the same edge twice in a row then that edge has the highest recalculated score in the whole queue and is added.

Data: $\mathcal{M}$, $\mathcal{D}$, C
Result: $\mathcal{R}' \subseteq \mathcal{R}_{\mathrm{C}}$
$\mathcal{R}' \leftarrow \{\}$;
foreach $d \in \mathcal{D}$ **do**
  $\mathcal{M}_d \leftarrow \{m \mid c_{m,d} = true\}$;
  foreach $m \in \mathcal{M}_d$ **do**
    $r \leftarrow (m, d)$;
    if `IsMaximalSource(`r`)` **or** `IsCarrier(`r`)` **then**
      $\mathcal{R}' \leftarrow \mathcal{R}' \cup \{r\}$;

Algorithm 3: Simple heuristic to reduce $|\mathcal{R}|$

3.5 Overall complexity analysis

We briefly describe the complexity of the different components of our method. Table 1 gives an example of running times on data generated as in Sec. 4.1.

Building the suffix tree requires $\mathcal{O}(n \log |\Sigma|)$ time, where n is the total length of text added and $|\Sigma|$ is the size of the alphabet. Preprocessing for LCA queries takes $\mathcal{O}(n)$ time, as does computing the word depths of the suffix tree nodes. We believe that identifying the valid n-grams and calculating their coverage takes $\mathcal{O}(n \log n)$ time although we have not proved this. Removing redundant n-grams takes $\mathcal{O}(n \log |\Sigma|)$. Computing the marker partial order takes $\mathcal{O}\left(|\mathcal{M}|^2 \log(|\Sigma|) \max_{d \in \mathcal{D}} |d|\right)$ time and in practice dominates the time complexity of the entire text reuse detection component. The heuristic described in Alg. 3 takes

$$\mathcal{O}\left(\frac{|\mathcal{R}_{\mathrm{C}}|^2}{|\mathcal{D}|} + |\mathcal{D}|\left(|\mathcal{M}| + |\mathcal{R}_{\mathrm{C}}|\right)\right)$$

expected time. Computing the partial order over $\mathcal{R}$ takes $\mathcal{O}\left(|\mathcal{R}|^2\right)$ time. An upper bound on the time complexity of the reconstruction method is $\mathcal{O}\left(|\mathcal{R}|^4\right)$; we hope to find tighter upper bounds on this.

4. EXPERIMENTAL RESULTS

In this section we demonstrate that using substring relations greatly improves the performance of an existing network inference method. That method, described in [8], is referred to as NetCover within this section. Our improved method is referred to as NetCover+substring.

4.1 Synthetic data

To demonstrate that taking the substring information into account significantly increases the accuracy of inferred net-

Table 1: Running times (in seconds)

Nb. of nodes	100	300	1000	3000
Build tree	7	24	42	92
Preprocess for LCA queries	3	5	8	22
Calculate word depths	1	1	2	3
Alg. 1	10	50	89	180
Remove redundant nodes	1	4	5	10
Compute substring partial order over $\mathcal{M}$	17	96	342	2691
Alg. 3 and $\mathcal{R}$ partial order	3	30	212	9818

works it is necessary to compare the networks obtained to ground truth networks. In the envisaged applications there is unlikely to be a "ground truth" (although in some circumstances there may be a surrogate ground truth network – *e.g.*, we can presume that the "follow" network in Twitter will be very strongly related to the network of "retweets"), so we use synthetic data where the ground truth is known to give a demonstration of the effectiveness of our method.

4.1.1 Generative model

We use a model that generates a directed network of outlets and then propagates a number of memes across the network using an infection model. We use two different random network generation methods, Erdős-Rényi [6] and Bollobás [4]. We choose parameter values for each such that the expected number of edges per node is 2 (expected 1 outgoing, 1 incoming). We let the number of nodes, N, be a parameter in the experiments.

To simulate the movement of memes through the system, we use a well known Susceptible-Infected-Recovered (SIR) model of infection, in which each outlet can be in one of three states. When an outlet is susceptible and one of its neighbours is infected there is a chance that the outlet will become infected. When an outlet is infected it generates a document carrying the infection. The way in which outlets change state is similar to as described in [8] except we set $p_R = 0$ and impose a time limit on how long an infection can propagate. We start M independent infections.

For simplicity the documents are generated with a fixed length, L, where each symbol, σ_i, is drawn uniformly from a finite integer alphabet, $\Sigma = \{0, 1, \ldots, |\Sigma| - 1\}$. The first outlet to be infected (the source) generates a fully random document. Subsequent outlets which become infected copy a random section of the document from which they become infected and pad the section with further random symbols to reach the length L, as shown in Fig. 3. The length of text which is copied is drawn from a uniform distribution with a parameter, α, determining the minimum amount of text to be copied, $l \sim \mathrm{U}(\alpha L, L)$. The starting position in the original document and the new document, x_0 and x_1 respectively, are drawn uniformly randomly and independently, $x_0, x_1 \sim \mathrm{U}(0, L - l)$.

This is obviously a crude model for how text is copied – in reality a multi-modal distribution is far more likely due to the likelihood of preserving sentence and paragraph structures when copying. Also we acknowledge that words in reality are not drawn uniformly from a finite alphabet, but rather from a Zipf power law. We do not, however, believe that these shortcomings in the generative model compromise the value of the experiments.

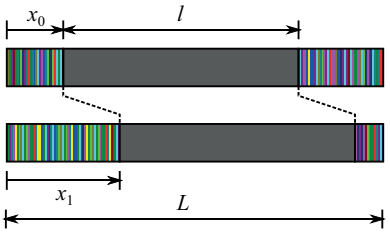

Figure 3: Generating a document when an outlet becomes infected.

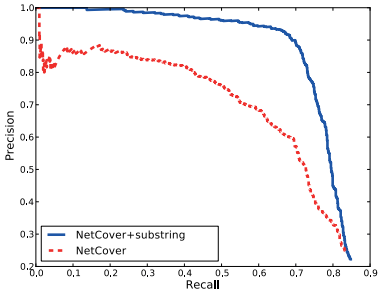

Figure 4: Precision/Recall for Erdős-Rényi graph

4.1.2 Results

We used the generative model to create one Erdős-Rényi graph and one Bollobás graph. Both graphs had $N = 500$ nodes. The Erdős-Rényi graph was used to generate documents for 300 markers and the Bollobás graph was used to generate documents for 1200 markers. The documents generated were processed directly as described in Sec. 3. Figs. 4 and 5 show how inference is dramatically improved by using substring information.

4.2 Real world data

We gathered data from the online mediasphere as described in Sec. 3.2 for four consecutive weeks from 3 January 2011. We chose constraint parameters for text reuse detection of $l = 100$ and $C_w = 2$.

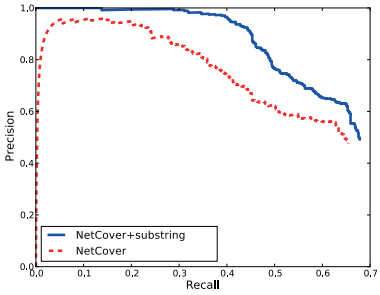

Figure 5: Precision/Recall for Bollobás graph

Table 2: Statistics of real world data

Week	1	2	3	4
Nb. of articles	121 360	128 280	124 830	124 729
Total length	57.2M	61.6M	59.5M	59.7M
$\lvert\mathcal{M}\rvert$	29 978	31 246	29 599	29 407
Articles with reports*	25 106	25 686	24 984	24 571
$\lvert\mathcal{R}_{\mathrm{C}}\rvert$	103 577	106 141	97 759	98 494
$\lvert\mathcal{R}'_{\mathrm{C}}\rvert$	92 481	94 981	87 393	88 311

* $\lvert\{d \mid \exists\,(m,d) \in \mathcal{R}_{\mathrm{C}}\}\rvert$

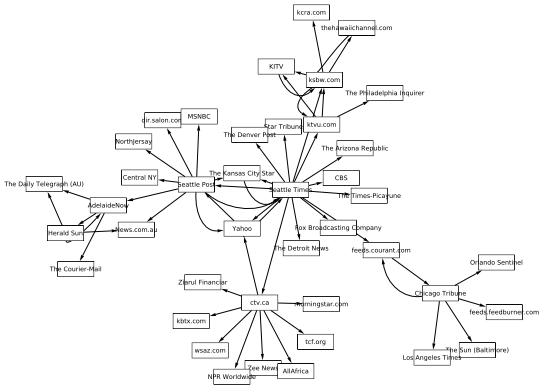

Figure 6: Largest connected component when 100 highest ranked edges are added to the reconstruction

4.2.1 Overview of results

Table 2 gives an overview of the data gathered and the results of detecting text reuse and generating candidate edges. Fig. 6 shows the largest connected component in the network inferred from Week 1 after 100 edges have been added, where a directed edge from A to B means the method inferred that information travelled from A to B. Further results can be found at the paper's supporting website [2].

We can see some interesting structure in the inferred network. For example, kcra.com, KITV and ksbw.com (all at the top of Fig. 6) are all Western and Pacific Region stations owned by Hearst Television Inc. The Daily Telegraph (AU), Herald Sun, news.com.au, Adelaide Now and The Courier-Mail (all on the left of Fig. 6) are all Australian newspapers owned by Rupert Murdoch's News Limited. The Chicago Tribune, feeds.courant.com (Hartford Courant), Orlando Sentinel, The Baltimore Sun, and Los Angeles Times (all in the bottom-right of Fig. 6) are all owned by the Tribune Company.

4.2.2 Stability of the network over time

As we do not have a ground truth network for this problem, we must demonstrate the value and the validity in a different way to in Sec. 4.1. We therefore test the stability of the network over time, as in [7].

The purpose of the test is to demonstrate that the inferred network for one week has much greater similarity to the inferred network for the previous week than would be expected by random chance. The effectiveness of this test is reliant on the underlying network itself being stable, which is an assumption of existing methods for network inference [9, 8].

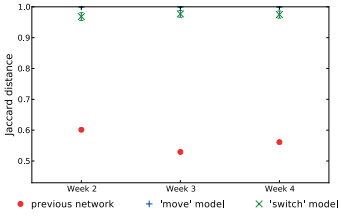

Figure 7: Network stability over time. Jaccard distance measures the distance between sets and is applied here to the set of edges.

To demonstrate that the network has greater similarity to the network inferred from data from the previous week than would be expected, we use two methods of randomization to produce random graphs preserving certain properties of the original network. The first method creates an Erdős-Rényi random graph with the same nodes and the same number of edges as the original network. We refer to this method as the "move model". The second method performs directed edge swaps to preserve the nodes, the number of edges and the in- and out-degree of the nodes. We refer to this method as the "switch model". We make $100 \times |E|$ swaps for each randomized network [7].

We repeated each randomization method 1000 times and the results are shown in Fig. 7. This figure shows much greater stability than would be expected by random chance and demonstrates that genuine results are being found. We also performed this test on networks generated by the original NetCover algorithm which also demonstrated greater stability than would be expected by random chance, but the Jaccard distances between networks inferred for consecutive weeks were slightly higher. This indicates that text reuse information is itself stable but that better inference is obtained using the substring information.

5. CONCLUSIONS

In this paper we have shown that existing methods for inferring networks of influence do not leverage a great deal of information which is often available and can greatly improve inference. We present an example of how we can greatly increase the effectiveness of the NetCover [8] algorithm applied to text reuse data by accounting for the way in which text is itself reused. Experiments on synthetic data showed very strong evidence for this claim. Such improvements should also be attainable by applying similar logic to other existing methods, such as NetInf [9].

We have also contributed a method for large scale text reuse detection based on the use of a suffix tree data structure, which can produce valuable observations of underlying network structures where text can be replicated.

We believe that the concept of leveraging the textual information present in observations on the Internet is a powerful one. We expect further work to identify more ways that textual information can indicate the directionality of relations between outlets.

Acknowledgements

Tristan Snowsill and Nick Fyson are supported by EPSRC grants DTG/EMAT.SB1848.6525 and EP/5011214 respectively. Nello Cristianini is supported by a Royal Society Wolfson Merit Award. This work is partially supported by EPSRC grant EP/G056447/1 amd by the European Commission through the PASCAL2 Network of Excellence (FP7-216866).

6. REFERENCES

[1] Spinn3r API. http://www.spinn3r.com/, 2008.

[2] Supporting website, https://patterns.enm.bris.ac.uk/projects/refining_causality, 2011.

[3] E. Adar and L. Adamic. Tracking information epidemics in blogspace. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 207–214. Citeseer, 2005.

[4] B. Bollobás, C. Borgs, J. Chayes, and O. Riordan. Directed scale-free graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 132–139. Society for Industrial and Applied Mathematics, 2003.

[5] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of operations research*, 4(3):233–235, 1979.

[6] P. Erdős and A. Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, pages 17–61. 1960.

[7] I. Flaounas, M. Turchi, T. De Bie, and N. Cristianini. Inference and validation of networks. In *Machine Learning and Knowledge Discovery in Databases*, volume 5781 of *Lecture Notes in Computer Science*, pages 344–358. Springer Berlin / Heidelberg, 2009.

[8] N. Fyson, T. De Bie, and N. Cristianini. Reconstruction of Causal Networks by Set Covering. In *ICANNGA'11: International Conference on Adaptive and Natural Computing Algorithms*, Ljubljana, Slovenia, 2011.

[9] M. Gomez-Rodriguez, J. Leskovec, and A. Krause. Inferring Networks of Diffusion and Influence. In *KDD2010: The 16th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2010.

[10] D. Gusfield. *Algorithms on strings, trees and sequences*. Cambridge University Press, 1997.

[11] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the Dynamics of the News Cycle. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 497–506. ACM New York, NY, USA, 2009.

[12] E. M. McCreight. A Space-Economical Suffix Tree Construction Algorithm. *Journal of the ACM*, 23(2):262–272, Apr. 1976.

[13] B. Schieber and U. Vishkin. On finding lowest common ancestors: simplification and parallelization. *SIAM Journal on Computing*, 17(6):1253–1262, 1988.

[14] P. Slavík. Improved performance of the greedy algorithm for partial cover. *Information Processing Letters*, 64(5):251–254, Dec. 1997.

[15] T. Snowsill and F. Nicart. A hard-disk based suffix tree implementation. Technical Report 133088, University of Bristol, Bristol, UK, 2011.

[16] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[17] P. Weiner. Linear pattern matching algorithms. In *IEEE Conference Record of 14th Annual Symposium on Switching and Automata Theory, 1973. SWAT'08*, pages 1–11, 1973.