A supervised approach to windowing detection on dynamic networks*

Benjamin Fish
University of Illinois at Chicago
1200 W. Harrison St.
Chicago, Illinois 60607
bfish3@uic.edu

Rajmonda S. Caceres
MIT Lincoln Laboratory
244 Wood St.
Lexington, Massachussetts 02421
rajmonda.caceres@ll.mit.edu

ABSTRACT

For any stream of time-stamped edges that form a dynamic network, a necessary and important choice is the aggregation granularity that an analyst uses to bin the data at. While this choice is often picked by hand, or left up to the technology that is collecting the data, the choice can make a big difference in the properties of the network. We call this the windowing detection problem. In previous work, this problem is often solved with a heuristic as an unsupervised task. As an unsupervised problem, it is difficult to measure how well a windowing algorithm performs. In addition, we show that the best windowing is dependent on which task an analyst want to perform on the network after windowing, and that therefore the task should be taken into account. We introduce a framework that tackles both of these issues: By measuring the performance of the windowing algorithm based on how well a given task is accomplished on the resulting network, we are for the first time able to directly compare different windowing algorithms to each other. Using this framework, we introduce windowing algorithms that take a supervised approach: they leverage ground truth on training data to find a good windowing of the test data. We compare the supervised approach to previous approaches and several baselines on real data.

ACM Reference format:
Benjamin Fish and Rajmonda S. Caceres. 2016. A supervised approach to windowing detection on dynamic networks. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 10 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Much big data mining on social and other types of networks either requires information about dynamics or is improved by such information. Incorporating temporal information can improve the efficacy and lead to more detailed analyses.

As data collection becomes cheaper and easier, the rate at which the data is being collected is often orders of magnitude more frequent than the underlying system itself is changing. The rate of the data collection process is typically a function of the technology used and not necessarily related to the evolution or dynamics of the network itself. Thus the data collection process makes a choice about the bin size - also called the *resolution*, *aggregation granularity*, or *time scale* of the dynamic network - that may not be the correct choice - binning the data at courser resolutions may make it possible to distinguish between noisy local temporal orderings and critical temporal orderings in the network. Indeed, the time scale of the network strongly impacts what structures and dynamics may be observed in the network [3, 12, 17]. Moreover, the choice of time scale impacts the efficacy of data mining on networks [8]. Thus for any data mining task over dynamic networks, choosing the bin size is not only important, but a necessary choice the data scientist must make, and leaving it up to the data collection process is not ideal. This is the problem we take up in this paper, variously called generating graph snapshots, oversampling correction, time scale detection, temporal scale inference, aggregation granularity detection, or windowing detection. We will refer to it as *windowing*, to emphasize the fact that the bin sizes - or windows - may not necessarily all be the same size.

This problem is marginally related to change point detection, which is the problem of detecting when the network changes drastically. What counts as 'drastic,' is often poorly defined, and is only a matter of degrees away from time scale detection, which asks for a segmentation of the sequence. Implicitly, however, change point detection assumes the network is observed at the right time scale. More generally, windowing is related to graph summarization tasks.

Typically, the windowing problem is framed as a unsupervised task. If the goal is merely data exploration and it is unclear what analysis we would wish to apply, then unsupervised windowing may be sufficient. On the other end of the spectrum, if sufficient knowledge of the data is at hand, the time scale may be chosen manually to represent natural scales, such as the diurnal, weekly, or monthly scales natural for dynamics among people. However, frequently, getting such domain-knowledge through a data exploration phase or otherwise may be prohibitively difficult or expensive. Moreover, we will demonstrate that the best windowing often cannot be defined independently of the analytical task.

In this paper, we show that the time scale for a dynamic network depends on the task - the most appropriate choice of scale for predicting new links appearing in the network

* This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.

may be different than the appropriate choice for detecting change points in the network, for example. The intuition is that the information needed to predict new links is different than the information needed to detect change points. This insight may appear quite intuitive, but it runs contrary to approaches for problems typically framed as unsupervised.

This informs our approach towards windowing: we set up the windowing problem as a supervised machine learning task. For example, for detecting change points, we set aside earlier training data from the network with labeled change points. We then find the right time scale for this labeled data, and apply the time scale to new data to find the change points there. In other words, the time scale detection takes as input the desired task, such as change point detection, and finds the right time scale for that task on the input data. Of course, the downside to this approach is that it requires training data, such as labeled change points on past data. However, we regard this as both a natural assumption and a necessary assumption: Many popular prediction and classification problems have some notion of ground truth. In addition, since the best time scale depends on the task anyway, we might as well tailor our time scale detection towards a particular goal.

In this paper, we use three different tasks as examples of goals: link prediction, attribute prediction, and change point detection. We use these three tasks as popular examples of tasks analysts perform on dynamic networks that have some notion of ground truth.

Our contributions are as follows: In addition to giving evidence that the best windowing of the data is task-dependent, we, for the first time, describe a simple framework for directly comparing the performance of windowing algorithms that leverages task-dependency. We also introduce windowing algorithms that takes a supervised-machine-learning approach. We compare using this approach against several baselines and previous work. We demonstrate that this supervised approach is often superior to other approaches, but that like all supervised approaches in machine learning, their quality may be dependent on the quality of the training data.

1.1 Previous Work

In numeric time series, this problem is often termed 'segmentation,' and segmentation of numeric times series has a long history which is outside the scope of this work; see [10] for an overview.

For dynamic networks, there has been some work related to our problem in the area of change point detection, which seeks to find points in time where the dynamic network has changed abruptly. Typical methods include using generative models of dynamic networks [16] or clustering like time slices [2]. This literature is marginally different from the problem addressed in this paper because - while finding change points do implicitly segment the dynamic network - the goal is not necessarily to find a good representation of the network for the purpose of binning each segment but rather just to find the points when the dynamic network undergoes significant change.

Also related is graph compression, which tries to find a representation of the dynamic network that minimizes the bits needed to store it while simultaneously retaining sufficient information about the network, under the general heading of graph summarization algorithms. This approach has been applied, more generally, to multi-layer networks, dynamic or not [6]. This problem is also slightly different from ours because we do not necessarily seek a small representation, merely an accurate representation for the task at hand. See [14] for a survey of graph summarization techniques.

Meanwhile, previous work on our problem has focused either on heuristic-based methods or methods that attempt to optimize for a specific metric on graphs. Caceres et al. maps the dynamic network to a time series using a metric on graphs (such as the number of triangles in each graph) and then determines time scale by finding the scale at which the time series' compression ratio and variance is balanced [21]. Soundarajan et al. determines a windowing of the data with respect to some metric (such as the exponent of the degree distribution) by measuring when that metric has converged [20]. In our view, these approaches have the downside that they require a specific metric. For a given data set or task to accomplish on a data set, it is not necessarily clear what metric to choose. Darst et al. use a parameter-free approach that seeks to find an appropriate time scale by measuring the similarity between graphs using the Jaccard index [5]. Most closely related to the approach that we take in this paper, Fish and Caceres use the quality of the performance of link prediction algorithms to determine the best time scale [8]. These are parameter-free methods or, relatedly, methods that assume that there is some 'ground-truth' time scale via a generative model or the like, as in [3, 5, 8]. In this paper, we do not take this tactic because as we demonstrate, choice of time scale may be dependent on the task at hand, which functions as a parameter for the problem.

1.2 Background

Consider a fixed set of vertices V and a dynamic network over V, represented as a stream of time-stamped edges. The goal of windowing is to segment this input stream of edges into (possibly overlapping) intervals to form a sequence of graphs $H_1, \ldots, H_m$, each graph (over V) representing all of the edges that occurred within each interval. Representing the input edge stream as a sequence of graphs $G_1,\ldots,G_t$, a *window* is an interval $G_i, G_{i+1}, \ldots, G_{i+k-1}$. Such an interval we refer to as a *window* and k the size of the window. A *windowing* is a sequence of windows $\{G_1, \ldots, G_{k_1}\}, \{G_{k_1}+1, G_{k_1}+2, \ldots G_{k_2}\}, \ldots, \{G_{k_{m-1}}, \ldots G_t\}$. In general, it is possible to also consider windows that overlap, but in this paper we focus on non-overlapping windows. The resulting sequence is $H_1, \ldots, H_m$, where H_i is the union of all edges that occurred in its window $G_{k_{i-1}}, \ldots, G_{k_i}$, i.e. $H_i = \cup_{j=k_{i-1}+1}^{k_i} G_j$. It is possible to consider other functions mapping a window to the resulting graph H_i, but we only consider the simple union in this paper. As a slight abuse of notation, in this paper we will refer to both the segmentation of the input graph sequence and the resulting sequence $H_1, \dots, H_m$ as a windowing.

If all windows have the same size w (except possibly the last window if the length of the sequence does not divide w), we refer to this as a *uniform windowing* and w the *bin size*, *window size*, or *time scale*. A window size of $w = 1$ represents the time scale of the collection process and a window size of $w = T$, where T is the duration of the observed network, means that all temporal information is ignored. The goal of this paper is to describe and evaluate windowing algorithms, i.e. algorithms for finding a windowing given an input sequence of graphs. As pointed out in [8, 20, 21], too small a window size may introduce noise and lack the structure necessary for analysis but too large a window size may lose important temporal information.

Once we have found a windowing $H_1, \dots, H_m$, it can now be the input for any task that operates over a dynamic network. We consider three popular tasks: link prediction, attribute prediction, and change point detection.

In this paper, we consider a supervised approach: the best windowing is the windowing that maximizes the performance of the algorithm for a given task, which we will refer to as simply the *task algorithm*. This gives us a way to compare different windowing algorithms: we may evaluate the performance of the windowing algorithm by evaluating the performance of the task algorithm on a test set. In general, we are given the edges of a dynamic network up to some time t as a training set, and performance is evaluated on the dynamic network from time $t + 1$ onwards. The windowing algorithm gets ground truth on the training set, e.g. the change points that occurred, and the task algorithm uses only the windowed graph sequence to conduct its analysis, say finding the change points in the test set.

2 TASKS

Given a candidate windowing, we evaluate its quality by performing a learning task algorithm on that windowing. We then evaluate the windowing by the performance of the task algorithm when using that windowing. We consider three task algorithms: link prediction, attribute prediction, and change point detection. We treat link prediction as an online task, and attribute prediction and change point detection as offline tasks. We do this to demonstrate windowing algorithms on both kinds of tasks. In what follows, we describe the algorithms we use for each task and how we judge their performances.

2.1 Link prediction

In link prediction, the goal is to predict the edges that are most likely to appear in the future. In the online setting, at every time step, our goal is to predict the edges that will appear in the next time step (the next step in the initial input sequence before windowing).

While there are many methods for link prediction (see [1] for a survey), the method we use is a simple scoring function that scores every pair of vertices by how likely an edge is to appear between them, typically using some notion of similarity. In this paper, we use the Katz$_\beta$ score, an efficient and well-performing score [13]. β is a damping parameter that weights shorter paths exponentially higher than longer paths. For our experiment results, we use $\beta = 0.005$, which has been used before [8, 13].

The performance of the link prediction algorithm is evaluated using the AUC of the precision-recall curve, as recommended by [23], averaged over all predictions made, one set of predictions for each graph.

2.2 Attribute prediction

We use the Time Varying Relational Classifier (TVRC) algorithm of Neville et al. [19] to determine the unknown value of a (binary) vertex attribute. As in their work, we assume attributes do not change over time. The goal is then to infer the missing attribute values by taking advantage of not only the known attribute values of the vertices but also the temporal information in the data. Their method is a modification of Naive Bayes: they build a probabilistic model of the likelihood of a vertex having a certain attribute value given the other values the vertex has, and the other values each neighboring vertex has, and assumes each of these are independent from each other. This model uses the time stamps to weight the influence each vertex has on its neighbors. The goal is then to find a windowing where TVRC builds the best performing model. TVRC requires a kernel for weighting the importance of edges - as they suggest, we use their exponential kernel $(1 - \theta)^{t-i}\theta$, where t is the total time, i the current time, and θ a hyperparameter controlling the rate of decay (for the sake of simplicity, we set $\theta = 1/2$ without trying to also optimize this parameter).

We use a special form of leave-one-out testing to measure the performance of TVRC. In this setting, the target attribute of one of the vertices is removed from both training and testing, a model is trained with the training set, and gives a prediction for the value of the missing attribute using the test set. To make the problem harder and more realistic, instead of just removing one target attribute, we remove a whole batch of them at once, and use the trained model to predict the values of all of their target attributes. The vertex set is partitioned into batches using a batch size parameter b, and this is repeated for each batch. Once a prediction has been made for all batches, we measure the performance or TVRC as the standard AUC of the ROC curve.

2.3 Change point detection

We use Graphscope, from Sun et al. [22]. Graphscope detects change points by estimating the times where segmenting the graph sequence at those times maximizes compressibility. Since our graphs are not bipartite, we make the necessary modifications to Graphscope so that it may be used in the non-bipartite case.

In order to evaluate a change point detection algorithm, we need a single score summing up how good a set of change points are, but to the best of our knowledge no such measure has been proposed in the literature, contrary to classification tasks, which frequently use AUC or other single values to summarize quality. We will need a single score rather than, say, a precision-recall curve, not only to evaluate, but also because our supervised windowing algorithm will need a single score to directly and automatically compare the quality of different window sizes.

To rectify this, we propose the following measure: Let $t_1, \ldots, t_k$ be the times of the ground-truth events, and $s_1, \ldots, s_\ell$ the times of the proposed events. Let n be the length of the sequence and $\delta(x)$ the function that equals 1 if x is true and 0 otherwise. Peel and Clauset [16] propose the following notions of precision and recall:

$$\text{Precision}(d) = \frac{1}{\ell} \sum_i \delta(\inf_j |s_i - t_j| \le d)$$

$$\text{Recall}(d) = \frac{1}{k} \sum_j \delta(\inf_i |s_i - t_j| \le d).$$

We will define the AUC of the precision-recall curve as the normalized volume under the curve given by Precision(d) and Recall(d) as d goes from 0 to n. Consider the set of distances $\{|s_i - t_j| : i, j\} \cup \{0, n\}$, ordered from smallest to largest as $d_1 < \ldots < d_m$, so $d_1 = 0$ and $d_m = n$. Then PR-AUC is defined as

$$\frac{1}{n} \sum_{i=1}^{m-1} (d_{i+1} - d_i) \cdot \text{Precision}(d_i) \cdot \text{Recall}(d_i).$$

For the sake of completeness, if $k = 0$ or $\ell = 0$, we define the PR-AUC to be 0. Note this is a $[0, 1]$-valued measure.

3 TASK DEPENDENCE

Ideally, it would be nice to have just one windowing algorithm that performs well regardless of whether your goal is link prediction, attribute prediction, change point detection, or any other task. However, we demonstrate that this does not appear to be feasible while still maximizing the performance of the task algorithm.

To do this, we score each window size by the performance of each of the three task algorithms when the data set is windowed at that size, so we have a score representing the quality of the window size for each of the three tasks. Tables 1 and 2 show the score for the ith task, if you chose the window size with the highest score for the jth task. For example, choose the best window size for link prediction. The two tables show that at that window size the other two tasks do not achieve as high a score as if you had chosen the best window size for those two tasks. (Each data set is broken up into consecutive intervals, and these scores are averaged over the scores for each interval as a form of k-fold validation, described in more detail in Section 4). This means that the windowing algorithm should choose a different window size for each of the tasks.

4 EXPERIMENTAL SETUP

In the offline setting, we set aside a previous interval of the dynamic network for training, and the next interval for testing. This interval includes ground-truth information, such as any change points that occurred in that interval. A uniform windowing algorithm may use this information to decide on a window size to use in the test set. A non-uniform windowing algorithm (or uniform windowing algorithm) is also allowed to see the edges in the test set to determine the windowing for the test set, but of course no ground truth information about the task. Once the test set is windowed, we perform our task on the windowed data and measure its performance, as describe in Section 2. The windowing algorithm's score is then just the score that the task algorithm received. For each data set, we split it up into six consecutive intervals, and then do training and testing on consecutive intervals, where the previous test set becomes the next training set, so there are five total tests (except for Reality Mining on change point detection where we split it into five instead of six intervals, in order to have every training set contain at least one change point). We use this form of k-fold validation is in order to promote generalizability of our results. For change point detection, we merely average the scores over each of the tests. For attribute prediction, we use pooling: we take the AUC as described in Section 2 over all vertices in all test sets, instead of averaging the individual AUC's of each test set, because the population, i.e. the vertices, is the same in each test set. When building the model for attribute prediction, we use only the first half of the training set to do this - this allows us to use the rest of the training data for testing the window sizes out, as described in more detail in Section 6.

In the online setting, a new graph from the initial input sequence is given to the windowing algorithm, and the windowing algorithm must make a decision as to how to incorporate the new graph into the windowing so far. After windowing, a prediction is made, and then the process is repeated. The score for the windowing algorithm is the average over all scores received for each prediction. As in the offline setting, we split each data set into six consecutive intervals and then do training and testing on each pair of intervals, where the testing phase is online.

5 DATA SETS

We use five data sets: Enron, MIT Reality Mining, Badge, Hypertext09, and Haggle. We treat all of these as undirected dynamic networks. For both convenience and uniformity, we bin each of these data sets at a initial window size at a 'natural' size, i.e. a choice that a data analyst might make, such as an hour or a day. This is so that a window size of $w = 1$ represents a baseline representing how well a hand-chosen windowing would perform. We only consider window sizes at least as large as this baseline.

Each of these are suitable for link prediction. Of these, Enron, Reality Mining, and Badge are equipped with vertex attributes in order to test attribute prediction, and of these,

Table 1: The score in the ith row and jth column is the score for the j task if the window size is chosen to maximize the score for the i task on the Enron data set. Scores are averaged over the different splits of the data.

	Link prediction	Attribute prediction	CP prediction
Link prediction	**0.188**	0.599	0.572
Attribute prediction	0.163	**0.649**	0.540
CP prediction	0.141	0.609	**0.935**

Table 2: The score in the ith row and jth column is the score for the j task if the window size is chosen to maximize the score for the i task on the Reality Mining data set. Scores are averaged over the different splits of the data.

	Link prediction	Attribute prediction	CP prediction
Link prediction	**0.277**	0.961	0.778
Attribute prediction	0.220	**0.983**	0.340
CP detection	0.258	0.961	**0.945**

Table 3: Spearman correlation coefficients and p-values for Enron

	Link prediction	Attribute prediction	CP detection
Link prediction	(1.0, 0.0)	(0.093, 0.005)	(-0.355, 4.19e-29)
Attribute prediction	(0.093, 0.005)	(1.0, 0.0)	(0.233, 5.11e-13)
CP detection	(-0.355, 4.19e-29)	(0.233, 5.11e-13)	(1.0, 0.0)

Table 4: Spearman correlation coefficients and p-values for Reality Mining

	Link prediction	Attribute prediction	CP prediction
Link prediction	(1.0, 0.0)	(-0.050, 0.493)	(-0.071, 0.330)
Attribute prediction	(-0.050, 0.493)	(1.0, 0.0)	(-0.188, 0.010)
CP detection	(-0.071, 0.330)	(-0.188, 0.010)	(1.0, 0.0)

Enron and Reality Mining also have established change point information.

5.1 Enron

This is an email network between employees of Enron Inc. from January 1999 to July 2002, during the period of Enron's market manipulation scandal and subsequent collapse [11]. We use the emails of 151 employees, where each edge is an email sent from one of those employees to another. Each vertex, representing an employee, have a binary attribute indicating whether the employee was a manager or not, and fifty integer-valued attributes, which are the number of occurrences in each employee's outgoing emails of the top fifty words used in all emails (a list of stop words were excluded from the top fifty words). The attribute we test on for attribute learning is whether the employee was a manager or not, which we took from [4]. We use the change points from [16], which represent times when the emails undergo substantial shifts, such as the launch of Enron online and changes in the CEO position. The initial bin size is one day.

5.2 Reality Mining

This is a proximity network of 90 MIT students and faculty using data taken from cell phones from September 2004 to May 2005 (we only use data up until the end of the academic year) [7]. We use as edges both phone calls between participants and whenever two participants are close to each other, detected using Bluetooth. Each of the participants filled out a survey about their cell phone usage, such as how much they use their cell phone, where they live etc., which we use as the categorical attributes for each vertex. The attribute we use for testing attribute prediction is whether they are part of the business school or the MIT Media Lab. Change points are taken from [16], which are the start and ends of vacations, semesters, etc. The initial bin size is one day.

5.3 Badge

This is a proximity network of 23 employees at a data server configuration firm for a month (the name of the data set comes from the badges the employees wore to track their location at the workplace) [15]. Each edge represents when two employees are in close proximity to each other, representing an interaction. Each employee was assigned a certain number of tasks, and data about these tasks was recorded, e.g. average completion time, whether they took on a difficult task or not, etc. For attribute prediction, we predict whether or not they made an error in one of their tasks. The initial bin size is one hour.

5.4 Haggle Infocomm

This is a proximity network consisting of interactions, recorded using Bluetooth, among attendees at an IEEE Infocomm conference over four days [18]. 41 attendees participated in this network. The initial bin size is 10 minutes.

5.5 Hypertext09

This is another proximity network of attendees at the ACM Hypertext 2009 conference, held over three days [9]. Each vertex is one of the 113 attendees, and an edge represents a interaction between two attendees that was active for at least 20 seconds. The initial bin size is 10 minutes.

6 WINDOWING ALGORITHMS AND BASELINES

We now describe the windowing algorithms that we will compare in both the offline and online settings. So that the supervised algorithms we introduce remain useful for not just the three tasks we consider, but any task, our algorithms do not attempt to take advantage of the particular nature of the task and corresponding algorithm at hand. In other words, we treat the task algorithms as black boxes. However, as we show in Section 7, the supervised approaches are still able to perform well despite this self-imposed constraint.

6.1 Offline supervised algorithms

Our intuition is very simple: since we know what task we want to accomplish on the test set, use the same task algorithm to learn the window size on the training set. In the offline setting, in order to try to prevent overfitting and to make the search space smaller, we only consider uniform windowings. This allows the algorithm to be very simple: For each window size, up to the length of the training set, window the training set at that size and use that as input for the task algorithm. Measure the performance of the task algorithm (remember we assume we have ground truth for the training set) and use that as the score for the window size. Window the test set with the window size that received the highest score. Of course, this means running the task algorithm $O(T)$ times (where T is the length of the training set), which is not particularly efficient. However, we make the assumption that since this is an offline setting, this blowup in running time in the training phase is not prohibitively large. We will refer to this as the offline *supervised* method.

Among the three tasks, the attribute prediction algorithm has an important difference, in so much as that it requires training data to build a model. We therefore need to make sure to decouple the training data for the model and the training data used to find the best window size. To do this, we split the training data into two, use the first half as the data for the model, and the second half of the training data to test out how well the model does when windowed at each window size. We do this by taking the vertices that still have the value of the target attribute and using the same process we use to test the quality of the attribute prediction on the test set: remove them in batches, build the model at each window size, and then test which value TVRC predicts on the second half of the training data. As described above, we then use the AUC as the quality of that window size.

Algorithm 1 Approximate windowing for link prediction

```
Parameters: M, H
  Initialize scores$_w$ as the empty list
  for each new graph $G_i$ do
    Let *new window sizes* include $w$ for $1 \leq w < i$ if length(scores$_w$) $< M$
    Let *best window sizes* include the top H window sizes by average(scores$_w$)
    for $w$ in *new window sizes*, *best window sizes* do
      Let $H_1, \ldots, H_{\lceil \frac{i-1}{w} \rceil}$ be the windowing of $G_1, \ldots, G_{i-1}$ at window size $w$
      predicted links = Katz($H_{\lceil \frac{i-1}{w} \rceil}$)
      Append AUC(new links in $G_i$, predicted links) to scores$_w$
    end for
    $w^* = \arg\max_w$ average(scores$_w$)
    Let $H_1, \ldots, H_{\lceil \frac{i}{w^*} \rceil}$ be the windowing of $G_1, \ldots, G_i$ at window size $w_{\text{best}}$
    return Katz($H_{\lceil \frac{i}{w^*} \rceil}$)
  end for
```

6.2 Online supervised algorithms

In the online setting, we could at each time step perform a similar procedure as in the offline case, leading to $O(i)$ runs of the task algorithm at the ith step, for a total of $O(T^2)$ times, where T is the total length of the sequence. This will often be prohibitively expensive. We introduce an approximate online windowing algorithm to deal with this issue. We illustrate with link prediction, as a task that is a natural example of an online task.

Every time we receive a new graph G_i, representing all the edges that occurred in the next time step, we can test each window size w by binning the sequence so far at size w and then use the last graph in the windowed sequence to predict the edges that will appear in G_i. We then compare the predicted edges to the actual edges in G_i, producing an AUC score for that window size[1]. The window size w^* chosen next to bin the sequence seen so far including G_i is the window size

[1]For the sake of computational efficiency, we only score pairs of vertices with non-zero degree, since the Katz score will produce a score of 0 for all other pairs.

Table 5: Performance of each of the algorithms on five data sets with respect to link prediction.

	Enron	Reality Mining	Badge	Hypertext	Haggle
Random	0.101	0.202	0.208	0.049	0.195
Hand-picked	0.148	**0.266**	**0.619**	**0.146**	**0.485**
Weighted Algorithm 1	**0.178**	**0.263**	**0.482**	**0.116**	0.443
Algorithm 1	0.153	0.259	0.428	0.100	**0.475**
Training only	0.159	0.240	0.418	0.065	0.437
ADAGE	0.149	0.198	0.394	0.044	0.298

Table 6: Performance of each of the algorithms on three data sets with respect to attribute prediction.

	Enron	Reality Mining	Badge
Random	0.566	0.966	0.583
Hand-picked	0.560	0.960	0.646
Supervised	**0.587**	**0.974**	**0.656**
No time	0.584	0.971	0.568
Fourier	0.567	0.967	0.568
Jaccard	0.576	0.973	0.571
Entropy	0.555	0.970	0.562
ADAGE	0.564	0.973	0.572

that maximizes the average of all scores for that window size so far. However, this still means testing $O(i)$ window sizes at each time step, for a total of $O(T^2)$ tests. To decrease the number of tests, we instead use an approximate version of this where only some of the window sizes are tested, as described in Algorithm 1[2]. Given hyperparameters to the algorithm M and H, we test a window size if it has either been tested fewer than M times, or if the average score so far ranks it amongst the top H window sizes. The intuition behind this approach is that a window size that has been performing badly will not suddenly become the best performing window size, and thus doesn't need to be tested. This requires only $O(M \cdot H \cdot T)$ total runs of the Katz algorithm instead of $O(T^2)$, where we think of as M and H as constants. In our experiments, we set $M = H = 10$ (We also demonstrate the effect of changing these hyperparameters in Section 7). We also test a weighted variant of Algorithm 1 by instead using a weighted average of the scores for each window size, in order to privilege scores closer to the present than the past. We use an exponential weighting scheme, where the weight for the score tested on the jth graph where t graphs have been seen already is $(1/2)^{t-j}$.

We also consider a version of this that stops generating new scores after the training period is over, and sticks with the best window size for the training data for all future time steps (*Training only*).

[2]This online algorithm is described explicitly for Katz, but it is worth nothing that the same approach may in principle be used for any online task.

6.3 Other windowing algorithms and baselines

We compare against *ADAGE*, the method of Soundarajan et al. [20]. ADAGE needs a metric as a parameter, so we use the exponent of the degree distribution, which is the metric they use. We also compare against the Jaccard-index-based method (*Jaccard*) of Darst et al. [5] and the entropy-based method (*Entropy*) of De Domenico et al. [6]. Since this method allows for graphs with any types of layers, we treat each time step as a layer and modify their method to only allow adjacent time steps to be merged.

We also compare against several baselines: the first is always using a window size of $w = 1$, which as mentioned above, represents a 'hand-chosen' value (which we will refer to as the *hand-chosen* algorithm). This represents as close to ground-truth as we can get on real data. The second baseline is the random algorithm (*Random*), which chooses a random windowing of the test set. In addition, for attribute prediction, we also consider the windowing that removes all temporal information, i.e. the window size that is always the length of the test sequence (*No time*). The final baseline we consider is slightly more sophisticated: Consider a time series of a graph sequence, one that assigns a real value to every graph in the sequence. For any such time series, we may compute its discrete Fourier transform (DFT). For frequency f, denote by x_f the amplitude of that frequency. The score we assign w is the maximum magnitude $|x_f|$ of any frequency in the transform such that f rounds to w. Only windows where there is such a frequency are assigned scores, and we choose the window size with the maximum score (the *Fourier* algorithm). In this paper, we consider the DFT under the commonly-used Hanning window and the time series we use is the number of edges in each graph. This serves as a proxy for the amount of activity at any given time. The DFT of this particular time series for dynamic networks has been used before, e.g. to study the Reality Mining data set [7].

All of these algorithms can be used in the offline setting, but only ADAGE, the hand-chosen baseline, and the random baseline can be used in the online setting.

7 RESULTS

Tables 4, 5, and 6 show our results. The supervised approach does do well, but there are certainly caveats. In link

Table 7: Performance of each of the algorithms on two data sets with respect to change point detection.

	Enron	Reality Mining
Random	0.660	0.609
Hand-picked	0.700	0.643
Supervised	0.649	0.490
Fourier	0.657	0.405
Jaccard	0.649	0.361
Entropy	0.709	**0.891**
ADAGE	**0.766**	0.356

prediction, the 'hand-picked' window size is often the best performer, probably due to the fact that these are well-studied data sets with natural periodicities dictated by human activity, and those window sizes reflect that. However, in general, we will want to have windowing algorithms that do not rely on an analyst's knowledge of the data set: acquiring domain-specific knowledge like this can be time-consuming, expensive, or difficult. Outside of this approach, at least one of the supervised methods does the best for all of the data sets. The weighted version is the overall winner, with a caveat: like any supervised technique, overfitting is always going to be an issue. We surmise that this is why the unweighted version outperforms the weighted version on the Haggle data set.

We also test the effect of the hyperparameters H and M, shown in Figure 2. The effect of increasing the hyperparameters appears noisy, especially on the Reality Mining data set. However, as would be expected increasing H or M, which increases the number of window sizes tested, generally increases performance. We chose small constant values of H and M to use for Algorithm 1, without attempting to optimize them, not only because we do relatively well without attempting any such optimization, but also because we consider the effect to be relatively weak, as the Reality Mining data shows.

In attribute prediction, the supervised approach is the clear winner, although the absolute difference over the others is sometimes rather small. On the other hand, the supervised approach does not do well for change point prediction. We believe this is due to a limitation of supervised approaches: they depend on the quality of the training data. Given the sparsity of change points, each training set contains only a very small number of change points, making it difficult to distinguish between different window sizes. This issue separates change point detection from our attribute prediction problem, which has a much greater amount of training data to work with. Figure 3 shows what happens as the amount of training data decreases for attribute prediction - i.e. the size of each batch of attribute values is removed increases. As training data decreases, so does the performance of TVRC under our supervised approach for windowing.

Our approach also reveals other differences between the three tasks. Figure 1 shows the quality of every window size on each of the first four intervals of Reality Mining (we can't use the last interval because each interval needs a subsequent

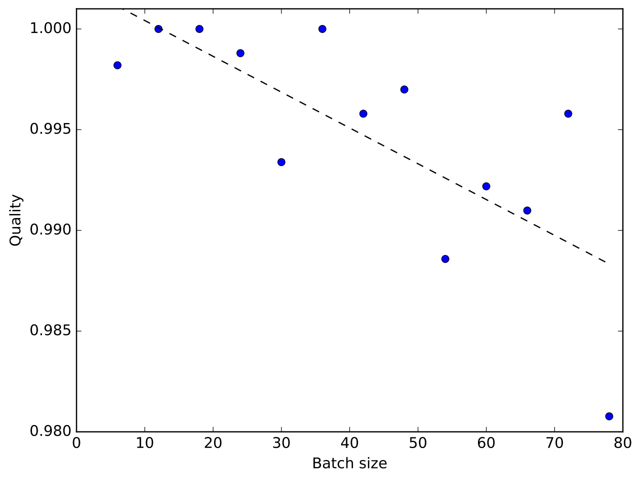

Figure 3: Performance of the attribute prediction algorithm on the Reality Mining data set as the batch size increases (where the batch size is the number of vertices whose target attribute has been removed before training and testing). Performance is averaged over the splits of the data. The dashed line indicates the least-squares linear regression.

interval for testing attribute prediction). The scores for change point detection are extremely sensitive to window size, with small differences in size making a large difference, indicating Graphscope's sensitivity to window size. TVRC, on the other hand, is much more stable under changes to window size, with link prediction falling somewhere in the middle. Such sensitivity makes it much more difficult to speed up the process by only testing some of the windows sizes instead of all of them, as we do here. We leave for future work determining if there is a way to test a fewer number of window sizes and the effect of choice of algorithm for a given task impacts sensitivity to window size.

Our supervised approach makes the basic assumption that a window size that works well in the past is more likely to work well in the future. To measure this, Figure 4 shows the difference between the two scores a window size got on consecutive intervals, averaged over all windows and consecutive intervals. Smaller differences mean that past data reflects future data more precisely. Figure 4 shows the difference was significantly larger for change point detection than the other two tasks, indicating that this assumption (at least when using Graphscope) is violated more under this task. This serves as evidence for why the supervised approach does not perform as well on change point detection.

8 CONCLUSION

In this paper, we have given evidence that windowing is task-dependent, and that this can be leveraged to produce more accurate windowing algorithms. Recognizing this, we have provided a simple and easy-to-use framework for directly comparing the quality of windowing algorithms and moreover,

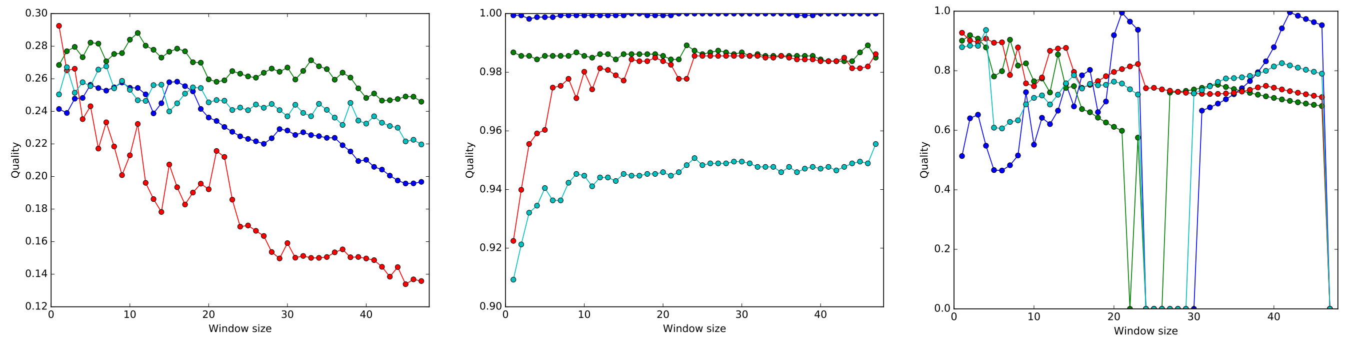

Figure 1: Results for each window size on each interval of the Reality Mining data. Each line corresponds with each of the four intervals. The left figure shows the quality of each window size for link prediction, the middle for attribute prediction, and the right for change point prediction.

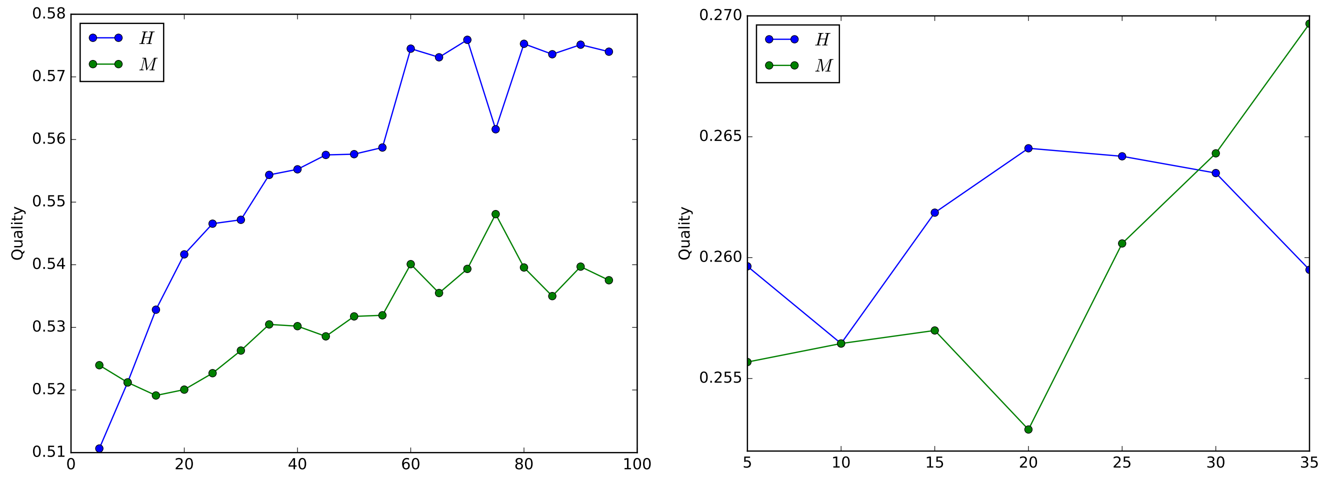

Figure 2: The effect of the hyperparameters H and M on quality for link prediction, where we fix H=10 and vary M, and fix $M = 10$ and vary H. The figure on the left is the results on the Badge dat set, while the right is the Reality Mining dataset.

introduced windowing algorithms that leverage our ability to test a windowing. Nonetheless, we leave for future work several challenges: Like any supervised machine learning, the quality of the learner depends on the quantity and quality of training data. Improving windowing algorithms in the face of environments with little training data remains an issue. Even with training data, using that training data to window can be a computationally-expensive procedure if the windowing algorithm has to repeatedly invoke an expensive task algorithm. We leave for future work finding heuristics that approximate well how a windowing will perform for a given task.

REFERENCES

[1] Mohammad Al Hasan and Mohammed J. Zaki. 2011. A survey of link prediction in social networks. In *Social Network Data Analytics*. Springer, 243–275.
[2] Michele Berlingerio, Michele Coscia, Fosca Giannotti, Anna Monreale, and Dino Pedreschi. 2010. As time goes by: Discovering eras in evolving social networks. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 81–90.
[3] Aaron Clauset and Nathan Eagle. 2007. Persistence and periodicity in a dynamic proximity network. *DIMACS Workshop on Computational Methods for Dynamic Interaction Networks* (2007).
[4] Germán Creamer, Ryan Rowe, Shlomo Hershkop, and Salvatore J Stolfo. 2009. Segmentation and automated social hierarchy detection through email network analysis. In *Advances in web mining and web usage analysis*. Springer, 40–58.
[5] Richard K Darst, Clara Granell, Alex Arenas, Sergio Gómez, Jari Saramäki, and Santo Fortunato. 2016. Detection of timescales in evolving complex systems. *arXiv preprint arXiv:1604.00758* (2016).
[6] Manlio De Domenico, Vincenzo Nicosia, Alexandre Arenas, and Vito Latora. 2015. Structural reducibility of multilayer networks. *Nature communications* 6 (2015).
[7] Nathan Eagle and Alex Sandy Pentland. 2006. Reality mining: sensing complex social systems. *Personal and ubiquitous computing* 10, 4 (2006), 255–268.
[8] Benjamin Fish and Rajmonda S Caceres. 2015. Handling oversampling in dynamic networks using link prediction. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 671–686.

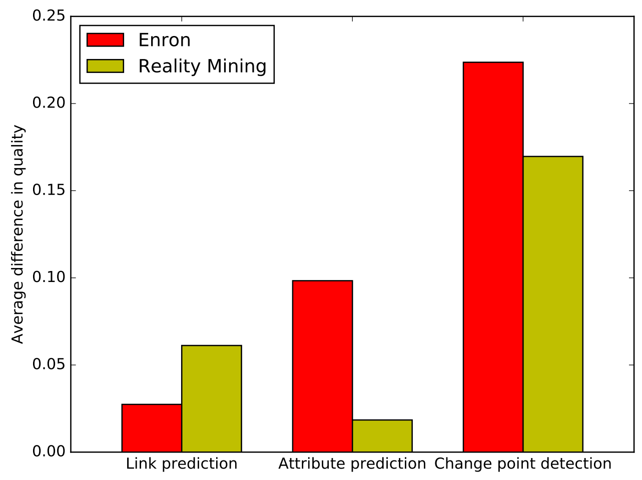

Figure 4: For each task, the average difference between quality of the same window size on consecutive intervals of both data sets.

[9] Lorenzo Isella, Juliette Stehlé, Alain Barrat, Ciro Cattuto, Jean-François Pinton, and Wouter Van den Broeck. 2011. What's in a crowd? Analysis of face-to-face behavioral networks. *Journal of theoretical biology* 271, 1 (2011), 166–180.

[10] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. 2004. Segmenting time series: A survey and novel approach. *Data mining in time series databases* 57 (2004), 1–22.

[11] Bryan Klimt and Yiming Yang. 2004. The enron corpus: A new dataset for email classification research. In *European Conference on Machine Learning*. Springer, 217–226.

[12] Gautier Krings, Márton Karsai, Sebastian Bernhardsson, Vincent D Blondel, and Jari Saramäki. 2012. Effects of time window size and placement on the structure of an aggregated communication network. *EPJ Data Science* 1, 1 (2012), 4.

[13] David Liben-Nowell and Jon Kleinberg. 2003. The Link Prediction Problem for Social Networks. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management (CIKM '03)*. ACM, New York, NY, USA, 556–559. DOI: http://dx.doi.org/10.1145/956863.956972

[14] Yike Liu, Abhilash Dighe, Tara Safavi, and Danai Koutra. 2016. A Graph Summarization: A Survey. *arXiv preprint arXiv:1612.04883* (2016).

[15] Daniel Olguín Olguín, Benjamin N Waber, Taemie Kim, Akshay Mohan, Koji Ara, and Alex Pentland. 2009. Sensible organizations: Technology and methodology for automatically measuring organizational behavior. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39, 1 (2009), 43–55.

[16] Leto Peel and Aaron Clauset. 2015. Detecting Change Points in the Large-Scale Structure of Evolving Networks. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[17] Bruno Ribeiro, Nicola Perra, and Andrea Baronchelli. 2013. Quantifying the effect of temporal resolution on time-varying networks. *Scientific Reports* 3 (2013), 3006.

[18] James Scott, Richard Gass, Jon Crowcroft, Pan Hui, Christophe Diot, and Augustin Chaintreau. 2009. CRAWDAD dataset cambridge/haggle (v. 2009-05-29). Downloaded from http://crawdad.org/cambridge/haggle/20090529. (May 2009). DOI: http://dx.doi.org/10.15783/C70011

[19] Umang Sharan and Jennifer Neville. 2008. Temporal-relational classifiers for prediction in evolving domains. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 540–549.

[20] Sucheta Soundarajan, Acar Tamersoy, Elias B Khalil, Tina Eliassi-Rad, Duen Horng Chau, Brian Gallagher, and Kevin Roundy. 2016. Generating Graph Snapshots from Streaming Edge Data. In *Proceedings of the 25th International Conference Companion on World Wide Web*. International World Wide Web Conferences Steering Committee, 109–110.

[21] Rajmonda Sulo, Tanya Berger-Wolf, and Robert Grossman. 2010. Meaningful selection of temporal resolution for dynamic networks. In *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*. ACM, 127–136.

[22] Jimeng Sun, Christos Faloutsos, Spiros Papadimitriou, and Philip S Yu. 2007. Graphscope: parameter-free mining of large time-evolving graphs. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 687–696.

[23] Yang Yang, Ryan N Lichtenwalter, and Nitesh V Chawla. 2015. Evaluating link prediction methods. *Knowledge and Information Systems* 45, 3 (2015), 751–782.