Joint Subgraph-to-Subgraph Transitions

Generalizing Triadic Closure for Powerful and Interpretable Graph Modeling

Justus Isaiah Hibshman
jhibshma@nd.edu
University of Notre Dame
Notre Dame, Indiana

Daniel Gonzalez*
dgonza26@nd.edu
University of Notre Dame
Notre Dame, Indiana

Satyaki Sikdar*
ssikdar@nd.edu
University of Notre Dame
Notre Dame, Indiana

Tim Weninger
tweninge@nd.edu
University of Notre Dame
Notre Dame, Indiana

ABSTRACT

We generalize triadic closure, along with previous generalizations of triadic closure, under an intuitive umbrella generalization: the Subgraph-to-Subgraph Transition (SST). We present algorithms and code to model graph evolution in terms of collections of these SSTs. We then use the SST framework to create link prediction models for both static and temporal, directed and undirected graphs which produce highly interpretable results. Quantitatively, our models match out-of-the-box performance of state of the art graph neural network models, thereby validating the correctness and meaningfulness of our interpretable results.

CCS CONCEPTS

• **Networks** → **Topology analysis and generation**.

KEYWORDS

graphs; subgraphs; triadic closure; link prediction

ACM Reference Format:
Justus Isaiah Hibshman, Daniel Gonzalez, Satyaki Sikdar, and Tim Weninger. 2021. Joint Subgraph-to-Subgraph Transitions: Generalizing Triadic Closure for Powerful and Interpretable Graph Modeling. In *Proceedings of the Fourteenth ACM International Conference on Web Search and Data Mining (WSDM '21), March 8–12, 2021, Virtual Event, Israel.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3437963.3441817

1 INTRODUCTION

Triadic closure is a widely known, simple process for modeling the evolution and dynamics of many real world graph processes [6, 14]. Triadic closure's use in the graph modeling community is due, in large part, to its ability to intuitively explain commonly observed social and natural phenomenon. For example, social balance theory is built upon achieving consistency among individuals in social network triads [8], and social networks commonly predict friendship links that close the most triangles [2]. In addition to triads, analysis of the evolution and dynamics of other small subgraphs (*i.e.*, graphlets, motifs, etc.) have proven to be illuminating and pleasantly interpretable for many graph mining and scientific tasks [17, 20, 22, 28].

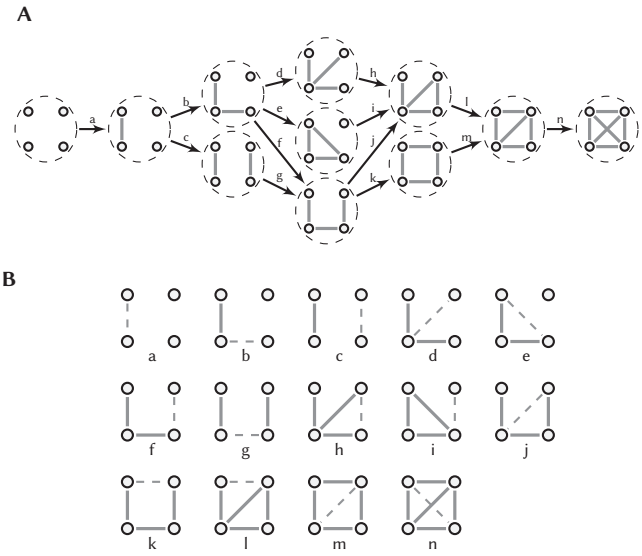

Figure 1: Edge-Addition Subgraph-to-Subgraph Transitions for Undirected Four-Node Subgraphs. The same information is depicted in two formats (*A* and *B*). *A:* Each arrow indicates a possible subgraph-to-subgraph transition (SST) caused by adding an edge to an undirected four-node subgraph. This format helps illustrate that subgraphs may transition to (and be transitioned to from) isomorphically distinct subgraphs. *B:* Each dashed missing edge indicates a possible subgraph-to-subgraph transition caused by adding the dashed edge to an undirected four-node subgraph. This format helps illustrate that each distinct edge-addition transition corresponds to an isomorphically-distinct missing edge in a subgraph.

*Both marked authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WSDM '21, March 8–12, 2021, Virtual Event, Israel
© 2021 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-8297-7/21/03…$15.00
https://doi.org/10.1145/3437963.3441817

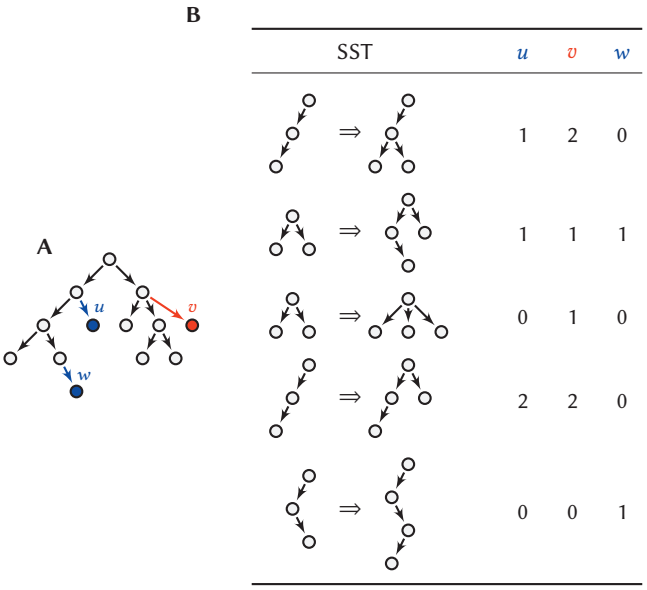

Figure 2: Growing a Binary Tree. The graph on the left (A) illustrates adding three new nodes to a binary tree, where each new node is connected by a single edge. The table on the right (B) enumerates the connected 3-to-4-node subgraph-to-subgraph transitions (SSTs) caused by adding the new nodes. Columns u, v, and w depict the number of SSTs in which each corresponding node in (A) is present.

To this end, researchers have generalized the concept of triadic closure in different ways. For instance, Seshadhri et. al. considered the many different kinds of triangle closures possible in a directed graph [27]. Yin et. al. considered something similar to Seshadhri et. al., but did not include bidirected edges in their enumeration of directed triadic closure types [31]. Rossi et. al. considered "motif closures," whereby they mean any occurrence of a motif being formed by the adding of an edge [24].

We offer an elegant generalization which encapsulates and expands upon previous generalizations of triadic closure: The Subgraph-to-Subgraph Transition (SST). In our formulation triangle closure can be considered one specific kind of subgraph-to-subgraph transition: open-wedge to triangle. SSTs are also a generalization of "motif closures," as a motif closure only considers the resulting subgraph, not the beginning subgraph (see [24]); thus a single motif closure may correspond to many distinct SSTs.

For example, Fig. 1 depicts all of the possible four-node subgraph-to-subgraph transitions in two formats: (A) as state transitions, and (B) with added edges. Although not shown in Fig. 1, our SST algorithms can handle significantly larger subgraphs (albeit with runtime implications), over directed or undirected graphs, for both node and edge additions and deletions.

When using triadic closure to model graph evolution, the essential question is: "How many triangles would this new edge close?" Put in the language of SSTs, we would ask, "How many wedge-to-triangle transitions would this new edge cause?" Once we move beyond a single SST such as wedge-to-triangle and into the world of multiple SSTs, we can then observe that any change to a graph corresponds to a collection (i.e. a multiset) of SSTs. Consider the example of a growing binary tree illustrated in Fig. 2. Here the table (B) on the right shows the 3-to-4-node SSTs caused by the addition of a new node and edge. Each addition has its own "feature vector" of associated SST counts, *i.e.*, the number of SSTs in which each new node and edge is present. In this small example, it is quickly evident that node v's connection, which does not follow the binary-nature of the rest of the graph, has a distinct vector from nodes u and w. Considering these collections grants new power for graph modeling, because it grants an important multidimensionality.

The larger the subgraphs one considers for SSTs, the more information one acquires (information-theoretically - meaningful human comprehension may decrease). The largest possible "subgraph" to subgraph transition one could consider simply consists of the full old graph and the full new graph.

In the present work we define a model for SSTs on directed and undirected graphs both with and without node and/or edge properties. We provide several analyses to show that SSTs can be used to model graph evolution and static graphs in a variety of contexts simply by fitting linear SVM models to the SST count vectors. Notably, these models are very *interpretable*, yet they perform comparably to state of the art neural network models (with their default hyperparameters) on static and temporal link prediction tasks. We also demonstrate, via a short case study, SSTs intuitively modeling a known graph process.

All our code is available at https://github.com/SST-Author/Subgraph-Subgraph-Transitions.

2 PRELIMINARIES

Before we formally introduce our SST model we first introduce some preliminary notation. We define a graph in the usual way. A simple directed or undirected graph is defined as $G = (V, E)$, where V is the set of vertices ("nodes") and $E \subseteq V \times V$ is the set of connections ("edges"). We use the convention that in undirected graphs $(u, v) = (v, u)$.

Induced Subgraph. Given a graph $G = (V, E)$ and a set of nodes $S \subseteq V$, the induced subgraph $G(S)$ is the graph consisting only of the nodes in S. Formally, $G(S) = (S, E_{G(S)})$ where $E_{G(S)} = \{(u, v) \mid (u, v) \in E \wedge u, v \in S\}$.

Node and Edge Properties. A graph's nodes and/or edges may have certain values associated with them. For instance, if an edge indicates a road, it might have a speed limit value. In some cases these properties may be important in how to model the graph. In those cases, we redefine the graph as follows: Let $G = (V, E, p_V, p_E)$ be a graph with two property functions, p_V and p_E. Each property function maps a node/edge and a property name to a value.

Isomorphisms and Automorphism Orbits. Given two graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ an *isomorphism* is a bijection $f : V_1 \rightarrow V_2$ such that $(u, v) \in E_1 \leftrightarrow (f(u), f(v)) \in E_2$. That is, an isomorphism is a mapping of a graph's nodes to another graph's nodes in a way that lines up the structures exactly. An *automorphism* is simply an isomorphism of a graph with itself.

When using node and edge properties (*e.g.* $G_1 = (V_1, E_1, p_{V_1}, p_{E_1})$ and $G_2 = (V_2, E_2, p_{V_2}, p_{E_2})$), an isomorphism must also preserve property values. Formally, $\forall v \in V_1.\, p_{V_1}(v) = p_{V_2}(f(v))$ and $\forall (u,v) \in E_1.\, p_{E_1}((u,v)) = p_{E_2}((f(u), f(v)))$.

The *automorphism orbit* of a node $v \in V$ is the set of nodes to which v is equivalent under automorphism. Formally, $AO(v) = \{u \mid u \in V \wedge \exists \text{ automorphism } f.\, f(v) = u\}$. Edges can have automorphism orbits through a similar definition: $AO((u,v)) = \{(a,b) \mid (a,b) \in E \wedge \exists \text{ automorphism } f.\, f(u) = a \wedge f(v) = b\}$.

Finding automorphism orbits in a graph is frequently thought of in terms of matching or refining a set of "colors," where nodes in the same orbit are given the same color and the original input graph may arbitrarily require that nodes be put in separate orbits by giving them different colors [18, 19]. We use this idea in our model to find the automorphism orbits of nodes in SSTs with node and/or edge properties.

3 SUBGRAPH-TO-SUBGRAPH TRANSITIONS

A subgraph-to-subgraph transition T is defined to be a pair of "before" and "after" subgraphs:

$$T = (G_T = (V_T, E_T, p_{VT}, p_{ET}), G'_T = (V'_T, E'_T, p_{V'T}, p_{E'T}))$$

Thus, there are many, many possible subgraph-to-subgraph transitions (SSTs). In this work, we limit our analyses to incorporate SSTs meeting certain conditions.

Specifically, we focus on modeling edge additions to a graph. Thus we restrict ourselves to the SSTs that indicate the addition of an edge. Additionally, we require that an SST does not include a change in property values. (The only allowed property changes are for a new edge to receive a property value. All existing nodes' or edges' values remain unchanged in the context of the SST.) This restriction on properties is a way to simplify our model/analyses in this work, but conceptually, the SST generalization allows for changing property values as well. We discuss the ways we do allow changing property values during our graph modeling in Section 4.2.1. Lastly, we require that the "after" subgraph be connected.

Notably, the code we release along with this paper includes the ability to acquire SST information for edge deletions, node additions, and node deletions. However, only edge additions are studied in the present work.

3.1 Properties in SSTs

We use SSTs to create interpretable models of graph evolution. To do so, we associate changes to a graph with SSTs. If we allowed numeric property values, the number of distinct SSTs would explode combinatorically, thereby making interpretation more difficult. To address this issue, our modeling algorithms require that each property be treated as one of the following:

(1) **Class Trait:** A class trait is a property which has a (ideally small) set of possible class values, where no ordering on the values is required.
(2) **Rank Trait:** A rank trait is a property which requires that the possible values are totally ordered (e.g. numbers). When labeling an SST with rank traits, our modeling algorithm does not use the raw values of the property but rather for each SST, the nodes (or edges) are sorted and the ranks of the nodes (or edges) are used rather than the raw property values. So for example, instead of listing raw PageRank values of the nodes in an SST (*e.g.* PageRanks: $\langle 0.34, 0.12, 0.12, 0.025 \rangle$), the SST would be encoded just with the relative ordering of those values (*e.g.* "PageRank Ordering: $\langle 2, 1, 1, 0 \rangle$").

4 SST GRAPH MODEL

Given our formalism, we implement a graph model that encodes and uses SSTs to model graph evolution. This graph model has three distinct modules:

- A "Transition Labeler," which takes a before and after subgraph and produces a canonical (*i.e.* automorphism-invariant) label for that subgraph-to-subgraph transition.
- A "Transition Counter," which takes a change to a graph (an edge addition, edge deletion, node addition, or node deletion) and enumerates all the SSTs induced by that graph change.
- Interpretable Static and Temporal Link Predictors which make use of the Transition Counter information.

4.1 The Transition Labeler

To correctly identify a subgraph-to-subgraph transition, we need to label it in a way that maps all isomorphically equivalent SSTs to the same label (*i.e.* a "canonical label"). To do this, we use an adaption of the Weisfeiler-Lehman isomorphism algorithm [29], (a process also known as "Color Refinement") which allows node and edge properties to be incorporated as "colors" [5, 9]. This algorithm provides a canonical node ordering for the vertices involved in the SST. The Weisfeiler-Lehman algorithm is not a *complete* isomorphism algorithm, but it is guaranteed to work on up to 9-node graphs (SSTs) [15].

As discussed earlier, an SST can be thought of as consisting of a "before" subgraph and an "after" subgraph. At first glance, it may seem that to produce a label for an SST, we must compute distinct canonical labels for the before and after subgraphs and then combine the labels. However, as discussed in Section 3, we limit our algorithms to working with four kinds of graph changes: edge addition, edge deletion, node addition, and node deletion, and we do not include property value changes in our SSTs. Thus, each SST we work with can be described as a **single** subgraph where the added/deleted edge/node is uniquely marked (*i.e.* colored); for an example, revisit Figure 1.

Thus, at a high level, the transition labeler works as follows:

(1) Receive as input a graph $G = (V, E, p_V, p_E)$, a set of nodes $S \subseteq V$, and a node or edge x to be added to or deleted from the subgraph induced by S. In the case of a node addition, the edges by which the new node initially connects to the network must be included in G. Similarly, in the case of a node deletion, the edges incident to the deleted node must be included in G.
(2) Uniquely label (*i.e.* color) x.
(3) For any rank traits (see Section 3.1), temporarily replace the property values with the relative ranking of those values. For example, ("Node Degrees": $\langle 30, 4, 12, 4 \rangle$) would be replaced with ("Node Degree Ranks": $\langle 1, 3, 2, 3 \rangle$).

(4) Convert node and edge trait values into node and edge "colors." Each distinct "color" corresponds to a unique combination of trait values. This ensures that nodes (or edges) with different property values will be assigned to different automorphism orbits (see Section 2).
(5) Given the above coloring, perform canonical color refinement on $G(S)$ to obtain a canonical node ordering O [5, 9].
(6) Use O to serialize $G(S)$, coupled with the information denoting the added/deleted edge or node. This produces a canonical label string H.
(7) Hash string H to output a canonical numeric label.

The computational bottlenecks of the algorithm are sorting the values of any included rank traits and running the color refinement algorithm. If $n = |S|$, $m = E_{G(S)}$, j = the number of node traits (properties) and k = the number of edge traits (properties), the ordering of rank traits and other trait processing can be completed in $O(j\, n \log n + k\, m \log m)$. Similarly, using an algorithm developed by Berkholz et. al. which can produce a canonical stable coloring even for edge-colored graphs, the SST labeler can run its "augmented Weisfeiler-Lehman" in $O((n+m)\log n)$ time [5]. Note that our implementation of color refinement is simpler algorithmically but less efficient than Berkholz et. al.'s ($O(n^2)$), but typically n is small enough that the difference does not matter.

4.2 The Transition Counter

To model a graph change (*e.g.* an edge addition), we wish to acquire counts of *all* the SSTs of a given size induced by the change. For the kinds of changes we model (edge addition, edge deletion, node addition, node deletion) all the SSTs will involve a few special nodes and their surrounding regions - one special node in the case of a node addition/deletion (the added/deleted node), two in the case of an edge addition/deletion (the edge's endpoints). We first note the one or two nodes involved in all of the SSTs and then employ a technique known as "Reverse Search" to enumerate all k-node connected subgraphs involving those nodes, where k is the desired SST size [3]. Lastly, for each of these connected subgraphs, we apply our Transition Labeler to obtain a canonical label for the SST.

At present, we are unaware of any techniques to compute the SSTs more efficiently than enumeration. Complex combinatorial tricks allow computing of three, four, and five-node subgraphs in a graph rapidly [10, 21]. At first glance it may seem that a simple solution to avoid enumeration is to efficiently count the subgraphs before and then after the graph change. However, while this would certainly produce useful information, it would not directly produce SSTs, since to know the SST counts one must know *which* subgraphs turned into *which* subgraphs; recall from Figure 1 that one subgraph can often transition into multiple other subgraphs. Additionally, our model requires that SSTs be allowed to have node and edge property values, but the state-of-the-art subgraph counters operate on property-less graphs. Nonetheless we do expect that future researchers will create quick, combinatorial methods for counting SSTs with node and edge properties, and we hope this paper serves as the spark that ignites that project.

As it is, if we hold the SST subgraph size constant at a value k, the runtime of our Transition Counter is effectively equivalent to the number of enumerated k-node subgraphs around the changes.

4.2.1 Trait Updaters. Our Transition Labeler forces property values to be the same in both the "before" and "after" halves of an SST. However our modeling system can still accommodate changes in property values across time. These changes simply are not directly shown in the SSTs. The Transition Counter allows the user to define "Trait Updaters" which can update property values before a set of changes is applied, just before a change's collection of SSTs is given labels, just after a change's SSTs are labeled, and after a full set of changes is applied. These options provide great flexibility, which we utilize in our link predictors (Sections 4.3.1 and 4.3.2).

4.3 Interpretable Link Predictors

Finally, to demonstrate the power of SSTs, we use them to create interpretable link predictors. Link prediction via subgraphs has been discussed by Juszycyszyn et al [11], Abuoda et al [1], and Zhang et al [32]. Likewise, the topic of "temporal motifs" distinct from SSTs has been discussed in Liu et al's survey [17].

Our predictors train for link prediction by collecting vectors of SST counts which correspond to adding actual/positive edges from training samples and vectors of SST counts which correspond to adding randomly sampled non-edges; then we separate the edges' SST vectors from random non-edges' SST vectors with a simple linear SVM. A linear SVM is certainly not the optimal model for prediction accuracy, but even a simple linear SVM with SSTs as its features performs quite well and, importantly, provides a simple way to interpret its predictions: the unit vector which defines the hyperplane separating real edges from non-edges.

Each component of the direction vector corresponds to a distinct SST. The magnitude of the component indicates the relative importance of the SST in distinguishing between actual edges and randomly sampled non-edges. SSTs with positive component values indicate an edge is more likely to be real; SSTs with negative component values indicate an edge is more likely to be a randomly sampled non-edge. We provide examples of interpreting SVM output in the results section.

4.3.1 Static Link Predictor. A static link predictor is given a single graph and tries to predict which edges may be missing. While SSTs are implicitly designed to model evolving graphs, we can apply them to static graphs relatively easily. To do this, we imagine each edge in the graph as having been "just added" by some temporal process. That is, for each edge, we can ask the question, "What SSTs would be involved if this edge was *not* present and then was added?" The imagined temporal process we uncover can then predict missing edges in terms of which edges are "most likely to be added next."

Our code "trains" on every positive edge plus α times as many randomly-sampled non-edges. In our experiments we set $\alpha = 10$.

In a directed graph, SSTs naturally distinguish between the two endpoints of an added edge by the direction of the new edge. While distinguishing between the two vertices being joined is not necessary, it may add useful information. Thus, if the graph is undirected, we create a "trait updater" (see Section 4.2.1) to allow the SSTs to distinguish between the two nodes being connected; whenever the addition of an undirected edge (u, v) is about to have its associated SSTs counted, this trait updater compares the degrees of u and v and then marks them as being of "equal degree" or "higher/lesser" degree in order to distinguish them.

4.3.2 Temporal Link Predictor. A temporal link predictor operates over series of interactions (edges) with timestamps. In the context of the present work the interactions are allowed to repeat across timestamps. Our temporal link predictor uses a fraction of its training edges as the "base graph" and then computes counts for an equal number of true edges and randomly-sampled non-edges.

To make use of the fact that edges have timestamps and may repeat, we create two edge traits and corresponding trait updaters (see Sections 3.1 and 4.2.1) to reflect the *recency* and *frequency* of interactions.

The *recency* trait is a "class trait" (see Section 3.1) and indicates when an edge most recently occurred. It sorts edges into four categories based on whether the edge:

(1) has never occurred before ("never").
(2) last occurred in the previous timestamp ("newest").
(3) last occurred in the timestamp before the previous ("new").
(4) last occurred at least three timestamps ago ("old").

Similarly, the *frequency* trait is also a class trait that sorts edges into four categories based on whether the edge:

(1) has never occurred before ("0").
(2) has occurred once before ("1").
(3) has occurred twice before ("2").
(4) has occurred three or more times before ("3+").

These traits for edges allow the SSTs to carry meaning that is simultaneously structural and temporal.

In our temporal link prediction tests (Section 5.4), the training/validation data is bucketed into nine timestamps. During testing our model uses the first eight timestamps' worth of interactions as the "base graph" and then computes the SST vectors for the ninth timestamp. Just using the latest edges for training has a twofold benefit: The edges being trained on and the graph at time of training most closely resemble the edges/graph at test time, and using only the latest edges speeds up training.

5 RESULTS

5.1 Modeling Known Graph Generators

Before proceeding to show our models operating on real-world graphs, we offer the reader a "warm-up" by demonstrating the ability of three-node SSTs to capture the well-known preferential attachment graph generation process first introduced by Barabasi and Albert [4]. The preferential attachment model generates a graph by creating a new node and wiring it to m existing nodes, with higher odds of connecting to a node that already has many edges.

We generate an example preferential attachment graph with $n = 1000$ and $m = 2$ and run the temporal link predictor on the temporal sequence of edge additions. As discussed earlier (Section 4.3), the SVM yields weights for each SST, which we use to describe the importance of each SST to the link prediction task. From these SSTs we see that our model captures many key aspects of the preferential attachment process. We illustrate our predictor's top twelve SSTs in Figure 3, in which the two subgraphs of an SST are combined into a single subgraph where source and target nodes of the new edge are indicated by shaded nodes, and edge colors indicate their recency. A positive weight above the subgraph indicates that the SST is more likely to be associated with a real edge addition than a random edge addition. A negative weight indicates the opposite.

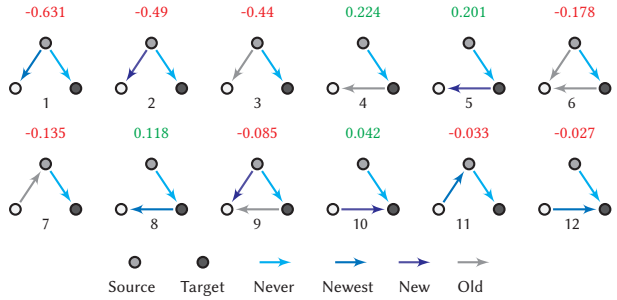

Figure 3: Top 12 3-Node SSTs from a preferential attachment graph process, listed in order of decreasing importance (see Section 4.3). SSTs are combined into a single subgraph where the new edge's source and target nodes are highlighted in gray and dark-gray respectively.

The results indicated in this example are in line with our expectations. The link predictor's top three most important SSTs along with SSTs 6 and 9 all indicate that a node cannot acquire out-edges at distinct timestamps. SSTs 4, 5, and 8 indicate that nodes are pointed-to only after they first point to other nodes, which is a key aspect of the preferential attachment process. Likewise, SST 10 suggests that a node has a higher chance of being pointed to if it was already pointed to recently. Similarly, SSTs 7 and 11 suggest that a node will not begin to point to another node after it has been pointed at. The ordering of SSTs 4, 5, and 8 indicates that newly-formed edges are more likely than randomly-sampled non-edges to point to nodes with older edges; this in turn indicates that nodes with older out-edges have more in-edges. Finally, the relative lack of triangles in the top 12 SSTs suggests that triangles are either rare, uninformative, or both.

These results demonstrate the interpretable power of SSTs to capture a well-known graph generation process which does not follow triadic closure. We now proceed to analyses of real-world graphs, generating both quantitative and interpretable results from the same model.

5.2 Quantitative Link Prediction Metrics

Many different metrics are used to quantitatively measure the link prediction performance. Some of the most common are the area under the ROC curve (i.e. "AUROC" or just "AUC"), and Hits at K.

However, Yang et. al. argue that AUC may not be a particularly meaningful metric for link prediction, and Hits at K can provide a very different picture depending on the selected K [30]. Instead, Yang et. al. demonstrate that area under the precision recall curve (AUPR) may be the best metric both in terms of what it represents and its discriminatory power. Ultimately a model with both high precision and high recall (and thus high AUPR) is of great use, but in an imbalanced setting like link prediction, a model with even a very small false positive *rate* and high true positive rate (and thus a high AUC) can still produce a high *number* of false positives compared to the number of true positives it produces, rendering its link predictions of little use in a real-world setting.

Unfortunately, for area under the precision recall curve (AUPR) to be meaningful, negative test cases must not be downsampled

[30]. However, having a model score every possible non-existent edge can be quite time-consuming. Thus, rather than reporting link prediction results for a whole graph, Yang et. al. recommend evaluating on the smaller task of link prediction between nodes a max distance of k apart, where k is some small number.

For comparability to other work, we report the AUC. For greater correctness, we report AUPR computed on the limited task of scoring all disconnected pairs of nodes (non-edges) within 3 hops and all connected node pairs (edges) that would be within 3 hops if they were to be disconnected. We call this "$AUPR_3$" to differentiate.

Properly Calculating AUPR Curve Areas. Area under the precision recall curve is often calculated via the trapezoidal rule, which effectively performs a linear interpolation between precision-recall points. This is incorrect, as explained by Davis and Goadrich [7], who introduce a superior interpolation in their seminal work. This difference becomes particularly relevant when models have large "gaps" in their precision recall curves.

5.3 Static Link Prediction

Next, we perform a quantitative and qualitative evaluation on three popular networks, detailed in Table 1, which are frequently used for static link prediction. Eu-core Emails is a correspondence graph from a European research institute where an edge indicates email(s) sent between two researchers. Cora ML and Citeseer are paper citation networks where edges indicate citations between papers.

Table 1: Datasets for link prediction.

	Dataset	Node Count	Edge Count	Temporal Edge Count
Static	Eu-core Emails	1,005	16,706	–
	Cora ML	2,708	5,278	–
	Citeseer	3,264	45,536	–
Temporal	Eu-core Temporal	986	24,929	332,334
	College Messages	1,899	20,296	59,835
	Wikipedia	100,312	746,086	1,627,472

We use an 85%/5%/10% split of the edges for training, validation, and testing respectively. Because there are no timestamps, the edges are partitioned randomly.

We report results for our models with both 3-node and 4-node SSTs. We compare against 2 baseline models, 4 state-of-the-art graph neural networks (GNNs) for undirected link prediction, and 1 state-of-the-art GNN for directed link prediction. For all the GNNs, we used the default hyperparameters from their source code.

Baseline Models. We defined two naive baseline methods: random and common neighbor count. The random baseline assigns edge predictions at random. The common neighbors method predicts that the more neighbors two nodes share in common, the more likely those two nodes are to connect [16]. Since the common neighbors heuristic does not directly apply to directed graphs, we count each of the four possible directed wedges connecting two nodes for directed graphs, similar to Yin et. al. [31].

Graph Variational Autoencoders. Graph Variational Autoencoders (GAEs) [13] have been recently developed to perform deep learning on graphs in support of tasks like link prediction and graph generation. GAEs are comprised of two parts. First an encoder that embeds a graph into a latent space by applying convolutional layers to an adjacency matrix. Second, using a simple inner-product decoder, GAEs produce an adjacency matrix of the same dimensions as the original input, which can be used for generating a new graph or for evaluating link prediction on the original graph.

Linear Variational Autoencoders. In response to the introduction of GAEs, Salha et. al. [25] questioned whether convolutional layers are really necessary for performing high-quality node embeddings. Their proposed Linear Variational Autoencoders (LinearAEs) replace the convolutional layers in GAEs with a simpler one-hop linear model which performs competitively on static link prediction. The overall behavior is similar to GAEs in that LinearAEs embed a graph's nodes and an inner-product decoder produces a new adjacency matrix for evaluation.

Gravity Graph Variational Autoencoders. A limitation of both GAEs and LinearAEs lies in their reliance on using inner products of vectors in the latent space for decoding. This imposes a strong restriction on the decoded adjacency matrices, which must always be symmetric. To circumvent this limitation, with the goal of performing directed link prediction, Salha et. al. [26] also introduced Gravity-Inspired Graph Variational Autoencoders (GravityAE), capable of generating non-symmetric adjacency matrices using a decoder based on taking sigmoid-activated logarithms of transformed latent vectors.

5.3.1 Quantitative Results. The undirected and directed static link prediction results are detailed in Tables 2 and 3 respectively. The SST-based models are consistently among the top performers.

It is important to note that the GNNs were trained with their default hyperparameters; no hyperparameter optimization was performed. This should make us take the GNNs' lower performance relative to our SST models' with a grain of salt, as our models have the advantage of requiring almost no hyperparameter tuning.

The key takeaway is *not* that our SST models will provide the best link prediction scores. Rather, the takeaway is that they provide *good* quantitative performance, *and thus our models' elegant and interpretable results are valid.*

5.3.2 Interpretation. As a case-study in the interpretability of SSTs on real-world graphs, we analyze the four-node SSTs from the Cora ML paper citation graph. Recall that the SVM effectively orders SSTs by how strongly they indicate that an edge is either a genuine edge or a randomly-sampled non-edge (Section 4.3).

We find that the SSTs ranked highest tend to involve bidirected edges (papers that cite each other, perhaps via pre-prints). Sometimes these SSTs are used to predict the presence/non-presence of bidirected edges; sometimes they simply use nearby bidirected edges as indicators of single-direction links. Predicting when a bidirected citation edge forms is a fascinating and difficult task but has limited applicability (Only 2.8% of connections in the Cora ML graph are bidirected.). Remember that the SVM ranks SSTs by how informative they are *if or when* they occur - *not* by how often they occur. Thus we look at both the SVM's top SSTs *with* bidirected edges and (to get a sense for how the SVM ranks more frequent SSTs) the top SSTs *without* bidirected edges. These are depicted and

Table 2: Link prediction performance on the static undirected graphs. The best and second-best performing models are boldfaced and underlined respectively.

Model	CiteSeer		Cora ML		Eu-core Emails	
	AUC	$AUPR_3$	AUC	$AUPR_3$	AUC	$AUPR_3$
CommonNeighbors	0.669±0.008	0.017±0.003	0.716±0.014	0.021±0.003	0.939±0.004	0.120±0.008
GCNAE	0.784±0.019	0.017±0.003	0.847±0.013	0.018±0.003	0.912±0.006	0.098±0.009
GCNVAE	0.788±0.015	0.016±0.002	0.846±0.011	0.017±0.003	0.904±0.008	0.090±0.013
LinearAE	0.775±0.014	0.019±0.003	0.829±0.014	0.021±0.003	0.923±0.005	0.120±0.005
LinearVAE	0.786±0.016	0.016±0.003	0.848±0.017	0.019±0.003	0.912±0.005	0.106±0.006
Random	0.491±0.019	0.004±0.000	0.496±0.018	0.002±0.000	0.500±0.012	0.004±0.000
SST-SVM-4	**0.865**±0.017	**0.020**±0.003	**0.879**±0.016	0.019±0.004	0.807±0.120	0.096±0.033
SST-SVM-3	0.754±0.016	0.019±0.002	0.823±0.011	**0.024**±0.003	**0.943**±0.004	**0.126**±0.008

Table 3: Link prediction performance on the static directed graphs. The best and second-best performing models are boldfaced and underlined respectively.

Model	CiteSeer (D)		Cora ML (D)		Eu-core Emails (D)	
	AUC	$AUPR_3$	AUC	$AUPR_3$	AUC	$AUPR_3$
CommonNeighbors	0.669±0.006	0.007±0.001	0.721±0.007	0.012±0.002	0.947±0.002	0.103±0.004
GravityGCNAE	0.500±0.013	0.002±0.000	0.506±0.015	0.001±0.000	0.657±0.018	0.004±0.000
GravityGCNVAE	0.516±0.008	0.002±0.000	0.512±0.011	0.001±0.000	0.826±0.004	0.008±0.000
Random	0.499±0.012	0.002±0.000	0.502±0.006	0.001±0.000	0.501±0.008	0.003±0.000
SST-SVM-4	**0.843**±0.01	**0.014**±0.003	0.877±0.011	0.011±0.002	0.886±0.039	0.137±0.002
SST-SVM-3	0.766±0.01	0.013±0.002	**0.891**±0.008	**0.018**±0.002	**0.970**±0.001	**0.176**±0.006

Table 4: Link prediction performance on the temporal directed graphs. The best and second-best performing models are boldfaced and underlined respectively.

Model	College Messages		Eu-core Temporal		Wikipedia	
	AUC	$AUPR_3$	AUC	$AUPR_3$	AUC	$AUPR_3$
CommonNeighbors	0.594±0.003	0.002±0.000	**0.938**±0.001	0.201±0.000	0.692±0.000	—
Random	0.499±0.007	0.001±0.000	0.498±0.004	0.008±0.000	0.500±0.001	—
TGN	0.749±0.000	**0.017**±0.007	0.762±0.014	0.005±0.000	—	—
SST-SVM-4	0.669±0.025	0.002±0.000	0.89±0.004	0.069±0.019	—	—
SST-SVM-3	**0.803**±0.010	0.008±0.002	0.933±0.003	**0.253**±0.020	**0.867**±0.001	—

analyzed in Figures 4 and 5 respectively. The SSTs pick up intuitive aspects of a citation network as well as some intriguing results.

5.4 Temporal Network Evolution

For evaluating networks' behavior over time, we perform future link prediction on three topologically rich, dynamic datasets, summarized in Table 1. Eu-core Temporal is a time-attributed version of the earlier Eu-core Emails dataset, incorporating timestamps on the emails. College Messages is a dynamic social network where edges indicate messages between users at certain times. Wikipedia is a temporal hyperlink network where the addition of a hyperlink from one page to another is represented by a timestamped edge.

For each network, the edges at time t indicate **interactions** at time t that can then be repeated at a later time.

Similar to the methodology used by Kasat et al, we bucket the interactions into τ evenly-sized buckets [12]. Since a bucket may cover multiple timestamps, an interaction (edge) may occur multiple times in a single bucket. We squash these multiple occurrences into a single edge and weight the interaction by its number of occurrences. In each bucket an edge's original timestamp is replaced with the index of that bucket. Thus we effectively have a series of τ graphs, $G_1, ..., G_\tau$. We train on the first $\tau - 1$ and test on G_τ. In our experiments we set $\tau = 10$. Note that neither our model nor the models we compare against make use of the weights; they just use the topology and the timestamps. However, if desired, one could add a "class trait" or "rank trait" (Section 3.1) to our temporal link predictor allowing it to make use of these values.

Temporal link predictors are fewer in number than their static counterparts. We compare against one state-of-the-art graph neural network and the baselines from before. As in our static evaluation, we test with both three-node SSTs and four-node SSTs.

5.4.1 Temporal Graph Neural Networks. As a state-of-the-art baseline for comparison on the task of temporal link prediction, we rely on the Temporal Graph Networks (TGNs) introduced by Rossi et. al. [23]. Their TGN is a graph autoencoder capable of temporally embedding a sequence of events on a graph (e.g., node additions

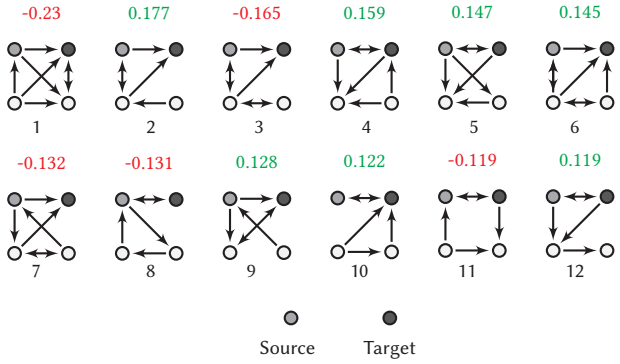

Figure 4: Top Cora SSTs with Bidirected Citations – (See Section 5.3.2) – SSTs 2, 3, and 6 indicate that if articles A and B mutually cite each other, A tends to cite whatever B cites *unless* another article C who bi-cites with B does not. SSTs 4 and 5 indicate that articles are more likely to bi-cite each other if they cite the same articles.

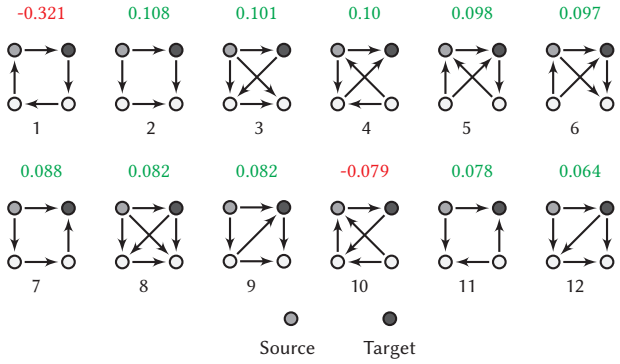

Figure 5: Top Cora ML SSTs Without Bidirected Citations – (See Section 5.3.2) – The top SST indicates that if an edge closes a 4-cycle that is considered a strong indicator that the edge is not genuine. Similarly, SST 10 suggests that a 3-cycle is unlikely, but not as unlikely as a 4-cycle. Other than SSTs 1 and 4, the top SSTs are positive indicators. SSTs 2 - 9, and 11 - 12 all include some kind of "transitivity", that nodes which cite (or are cited by) similar articles cite each other.

or deletions) using temporal graph attention layers. A Multi-Layer Perceptron decoder allows the TGN to score candidate edges with probabilities for evaluation of future link prediction.

5.4.2 Quantitative Results. Quantitative results are listed in Table 4. Once again our SST-based link predictors are among the top performers. Again, we suggest that these numbers be taken with a grain of salt because we simply used the GNNs' default hyperparameters. Chiefly, our tests demonstrate that our SSTs' elegant and interpretable results are validated by good prediction performance.

Note that we bypassed computing AUPR_3 on the Wikipedia graph due to the sheer size of the false test edge set - $O((10^5)^2)$.

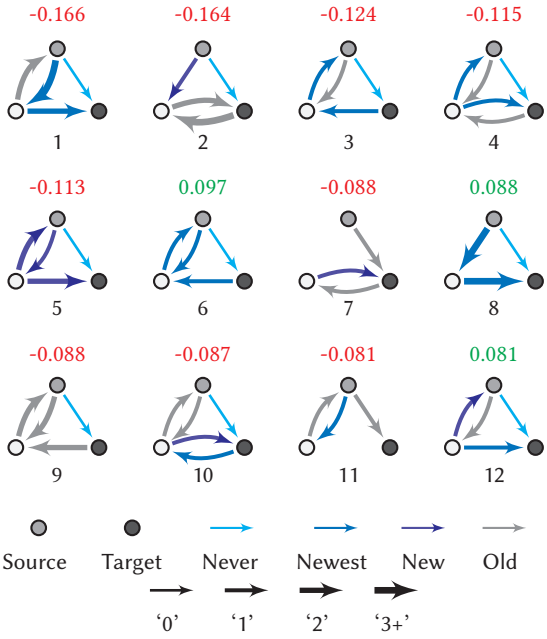

Figure 6: Top 3-Node SSTs for Wikipedia Link Additions – Main Takeaway: Wedges only close to triangles when the wedge had recent edges (*e.g.* Newest) appearing for the first or maybe second time (low frequency, *e.g.* '1'), ideally including an edge pointing to the target node of the new edge.

5.4.3 Interpretable Temporal Results. To demonstrate the interpretability of SSTs on temporal graphs, we explore the three-node SSTs on the Wikipedia edge additions graph. We find that, unlike the general assumption of triadic closure, according to our model many triangles are considered *unlikely* to close. It is only the triangles where certain connection combinations in the wedge were formed recently (indicated by our recency trait) and *for the first (or maybe second) time* (indicated by our frequency trait) that the wedge is quite likely to close into a triangle. See Figure 6. This is evidenced quantitatively by the fact that the three-node SST predictor performed much better than the Common Neighbors.

6 CONCLUSION

We defined an elegant generalization of Triadic Closure, the Subgraph-to-Subgraph Transition (SST). This generalization allowed us to use a simple classifier, the Linear SVM, to create interpretable link prediction models which performed comparatively with state of the art graph neural networks. We expect that the Subgraph-to-Subgraph Transition will become a standard tool in modeling graphs and that future research will produce new and creative ways to use and efficiently count SSTs.

ACKNOWLEDGEMENTS

This research is supported by a grant from the US National Science Foundation (#1652492).

REFERENCES

[1] Ghadeer AbuOda, Gianmarco De Francisci Morales, and Ashraf Aboulnaga. 2019. Link prediction via higher-order motif features. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 412–429.

[2] Lada A Adamic and Eytan Adar. 2003. Friends and neighbors on the web. *Social networks* 25, 3 (2003), 211–230.

[3] David Avis and Komei Fukuda. 1996. Reverse search for enumeration. *Discrete Applied Mathematics* 65, 1-3 (1996), 21–46.

[4] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.

[5] Christoph Berkholz, Paul Bonsma, and Martin Grohe. 2017. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory of Computing Systems* 60, 4 (2017), 581–614.

[6] Ginestra Bianconi, Richard K Darst, Jacopo Iacovacci, and Santo Fortunato. 2014. Triadic closure as a basic generating mechanism of communities in complex networks. *Physical Review E* 90, 4 (2014), 042806.

[7] Jesse Davis and Mark Goadrich. 2006. The relationship between Precision-Recall and ROC curves. In *Proceedings of the 23rd international conference on Machine learning*. 233–240.

[8] Mark S Granovetter. 1977. The strength of weak ties. In *Social networks*. Elsevier, 347–367.

[9] Martin Grohe, Kristian Kersting, Martin Mladenov, and Pascal Schweitzer. 2017. Color refinement and its applications. *Van den Broeck, G.; Kersting, K.; Natarajan, S* (2017), 30.

[10] Madhav Jha, C Seshadhri, and Ali Pinar. 2015. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the 24th International Conference on World Wide Web*. 495–505.

[11] Krzysztof Juszczyszyn, Katarzyna Musial, and Marcin Budka. 2011. Link prediction based on subgraph evolution in dynamic social networks. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*. IEEE, 27–34.

[12] Hemant Kasat, Sanket Markan, Manish Gupta, and Vikram Pudi. 2019. Temporal Link Prediction in Dynamic Networks. In *Proceedings of the Mining and Learning on Graphs workshop*.

[13] Thomas N Kipf and Max Welling. 2016. Variational Graph Auto-Encoders. *NIPS Workshop on Bayesian Deep Learning* (2016).

[14] Peter Klimek and Stefan Thurner. 2013. Triadic closure dynamics drives scaling laws in social multiplex networks. *New Journal of Physics* 15, 6 (2013), 063008.

[15] AA Leman. 1970. On automorphisms of certain classes of graphs. *Avtomatika i Telemehanika* 2 (1970), 75–82.

[16] David Liben-Nowell and Jon Kleinberg. 2007. The link-prediction problem for social networks. *Journal of the American society for information science and technology* 58, 7 (2007), 1019–1031.

[17] Penghang Liu, Valerio Guarrasi, and A Erdem Sarıyüce. 2020. Temporal Network Motifs: Models, Limitations, Evaluation. *arXiv preprint arXiv:2005.11817* (2020).

[18] Eugene M Luks. 1982. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of computer and system sciences* 25, 1 (1982), 42–65.

[19] Brendan D McKay and Adolfo Piperno. 2014. Practical graph isomorphism, II. *Journal of Symbolic Computation* 60 (2014), 94–112.

[20] Ashwin Paranjape, Austin R Benson, and Jure Leskovec. 2017. Motifs in temporal networks. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. 601–610.

[21] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. 2017. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web*. 1431–1440.

[22] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinformatics* 23, 2 (2007), e177–e183.

[23] Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. In *ICML 2020 Workshop on Graph Representation Learning*.

[24] Ryan A Rossi, Anup Rao, Sungchul Kim, Eunyee Koh, and Nesreen Ahmed. 2020. From Closing Triangles to Closing Higher-Order Motifs. In *Companion Proceedings of the Web Conference 2020*. 42–43.

[25] Guillaume Salha, Romain Hennequin, and Michalis Vazirgiannis. 2020. Simple and Effective Graph Autoencoders with One-Hop Linear Models. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*.

[26] Guillaume Salha, Stratis Limnios, Romain Hennequin, Viet Anh Tran, and Michalis Vazirgiannis. 2019. Gravity-Inspired Graph Autoencoders for Directed Link Prediction. In *ACM International Conference on Information and Knowledge Management (CIKM)*.

[27] C Seshadhri, Ali Pinar, Nurcan Durak, and Tamara G Kolda. 2017. Directed closure measures for networks with reciprocity. *Journal of Complex Networks* 5, 1 (2017), 32–47.

[28] Johan Ugander, Lars Backstrom, and Jon Kleinberg. 2013. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of the 22nd international conference on World Wide Web*. 1307–1318.

[29] Boris Weisfeiler and Andrei A Lehman. 1968. A reduction of a graph to a canonical form and an algebra arising during this reduction. *Nauchno-Technicheskaya Informatsia* 2, 9 (1968), 12–16.

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

[31] Hao Yin, Austin R Benson, and Johan Ugander. 2019. Measuring directed triadic closure with closure coefficients. *arXiv preprint arXiv:1905.10683* (2019).

[32] Muhan Zhang and Yixin Chen. 2017. Weisfeiler-lehman neural machine for link prediction. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 575–583.