Efficient Algorithms to Mine Maximal Span-Trusses From Temporal Graphs

Quintino F. Lotito
University of Trento
Trento, Italy
quintino.lotito@studenti.unitn.it

Alberto Montresor
University of Trento
Trento, Italy
alberto.montresor@unitn.it

ABSTRACT

Over the last decade, there has been an increasing interest in temporal graphs, pushed by a growing availability of temporally-annotated network data coming from social, biological and financial networks.

Despite the importance of analyzing complex temporal networks, there is a huge gap between the set of definitions, algorithms and tools available to study large static graphs and the ones available for temporal graphs.

An important task in temporal graph analysis is mining dense structures, i.e., identifying high-density subgraphs together with the span in which this high density is observed.

In this paper, we introduce the concept of (k, Δ)-truss (span-truss) in temporal graphs, a temporal generalization of the k-truss, in which k captures the information about the density and Δ captures the time span in which this density holds. We then propose novel and efficient algorithms to identify maximal span-trusses, namely the ones not dominated by any other span-truss neither in the order k nor in the interval Δ, and evaluate them on a number of public available datasets.

CCS CONCEPTS

• **Theory of computation** → **Graph algorithms analysis**; • **Mathematics of computing** → **Graph theory**; • **Information systems** → **Spatial-temporal systems**; *Data mining*; *Web searching and information discovery*.

KEYWORDS

Community detection, Dense structures, Graph mining, Social networks analysis, Temporal graphs

1 INTRODUCTION

Despite the fact that graph theory has been studied for centuries, in the last years there has been an explosion in the interest of the
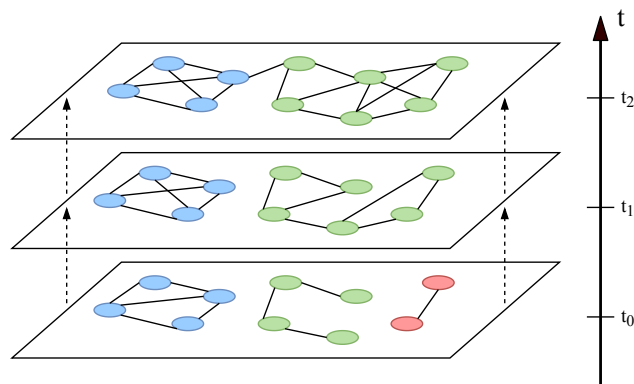
Figure 1: A temporal graph with time-evolving communities. It is represented as a sequence of static graphs; each static graph is a snapshot of the temporal graph at a certain time.

research community in network-related fields. This is mainly motivated by the increasing interest in social networks – which can be defined as a set of social entities (such as people, groups, and organizations) together with the relationships or interactions between them – and by a proliferating availability of network datasets coming from online social networks (e.g., Facebook, Twitter, Instagram, YouTube), biological networks (e.g., molecular interactions) or financial interactions.

So far, most of the work in social network analysis has focused on static graphs. The growing availability of temporally-annotated network data coming from social, biological and financial networks creates the opportunity to fill the gap between the set of definitions, algorithms and tools available for large static graphs, and the ones available to analyze temporal graphs. The latter are defined as graphs that change over time (i.e., whose edges are not continuously active). However, it is not yet clear how introducing the notion of time will affect the computational complexity of combinatorial graph problems [9].

Just to mention a few examples, temporal graph modelling and analysis of temporal properties can have applications in sociology and social network analysis (e.g., find voting patterns based on social media posts); security and distributed computing (e.g., design strategy to contain the spread of malware in computing devices); biology (e.g., study the set of chemical reactions that occur in a healthy organisms) [9].

A property of real-world graphs is that they tend to be globally sparse but locally dense, meaning that while the entire graph is sparse (i.e., vertices have a small average degree), it contains dense

subgraphs (i.e., groups of vertices with a large number of links among each other). In general, density is an indication of relevance. Dense regions in a network may indicate high degrees of interaction and mutual similarity. In real-world applications, these regions may indicate characteristics like attractive forces or favourable environments [11].

The enumeration of the dense components of a graph can either be the main goal of an analysis task, or act as a preprocessing step aiming to reduce the graph by removing sparse parts, in order to conduct more complex and time-consuming analysis [5].

A number of definitions of dense structures have been proposed in literature, ranging from cliques (i.e., subgraphs in which every vertex is adjacent to every other vertex), to some relaxations of the clique, such as k-cores, the k-trusses, or the k-plexes.

The previously mentioned concepts of dense structures can be generalized to the temporal case, in which one can be interested in mining high-density subgraphs together with the span in which this high density is observed. Having a set of tools to extract these structures enables a detailed comprehension of the network dynamics and can act as a building block towards more complex tasks and applications [7].

To name some examples of applications, we can rely on temporal dense structures computation to mine stories from social networks (i.e., events capturing popular attention in social media), which can be identified by finding a group of entities (i.e., people, locations, companies or products) strongly associated for a reasonable amount of time [2]; we can mine well-acquainted individuals from a collaboration network and form successful teams; we can analyze protein-interaction networks and locate protein complexes that are densely interacting at different states, indicating possible underlying regulatory mechanisms [19].

In this paper, we follow the approach of Galimberti et al. [7], who introduced the concept of the span-cores of a temporal graph (a temporal generalization of the k-core dense structure), and define the concept of (k, Δ)-*trusses* (*span-trusses*), a temporal generalization of the k-truss, in which k captures the information about the density and Δ captures the time span in which this density holds. We propose novel and efficient algorithms to discover the maximal span-trusses of a temporal graph, i.e., the ones not dominated by any other span-truss neither in the order k nor in the interval Δ.

We conclude the paper by evaluating our contributions on a number of public available real-world network datasets, showing that our proposals consistently outperform the baseline proposed for this task.

2   BACKGROUND

k-truss is a dense structure which considers the involvement between the structures of edges and triangles. It has been introduced based on the observation of social cohesion, where triangles play an essential role [6]. The k-truss community model has three significant advantages: strong guarantee on cohesive structure, few parameters and low computational cost [10].

Definition 2.1 (Triangle). Given a graph $G = (V, E)$, a *triangle* in G is a cycle of length 3.
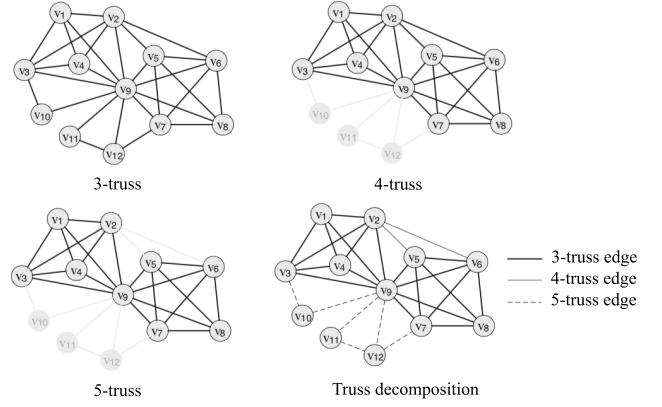

Figure 2: Example of the truss decomposition of a graph [5].

Definition 2.2 (Support of an edge). Given a graph $G = (V, E)$ and an edge $e \in E$, the *support* $\sup(e)$ is the number of triangles that e participates in.

Definition 2.3 (k-truss). Given a graph $G = (V, E)$, the *k-truss* of G, where $k \geq 2$, is defined as the largest subgraph g of G in which every edge is contained in at least $(k - 2)$ triangles within the subgraph, i.e., $\sup_g(e) \geq k - 2, \forall e \in g$.

It is easy to see that a k-truss is an *edge-induced subgraph*.

Definition 2.4 (Maximal k-truss). A k-truss T_k of a graph G is said to be *maximal* if there does not exist any other k-truss $T_{k'}$ such that $k' > k$.

Problem 2.1 (Truss decomposition). The problem of truss decomposition in a graph G is to find the (non-empty) k-trusses of G for all k [22].

Observation 2.1 (Containment). Each k-truss of a graph G is a subgraph of the $(k - 1)$-truss of G; for example, in 2, the 5-truss is a subgraph of the 4-truss which in turn is a subgraph of the 3-truss.

An algorithm to efficiently compute the truss decomposition of a static, unweighted, undirected graph $G = (V, E)$ has been proposed by Wang et al. [22]. This algorithm resorts to an in-memory triangle counting algorithm [16] and bin sort to achieve a complexity of $O(|E|^{1.5})$.

3   PROBLEM STATEMENT

We are given a temporal graph $G = (V, T, \tau)$, where V is a set of vertices, $T = [0, 1, ..., t_{max}] \subseteq \mathbb{N}$ is a discrete-time domain, and $\tau : V \times V \times T \rightarrow \{0, 1\}$ is a function defining for each pair of vertices $u, v \in V$ and each timestamp $t \in T$ whether edge (u, v) exists in t.

We denote $E = \{(u, v, t) \mid \tau(u, v, t) = 1\}$ the set of all temporal edges. Given a timestamp $t \in T$, the set of edges existing at time t is $E_t = \{(u, v) \mid \tau(u, v, t) = 1\}$.

A temporal interval $\Delta = [t_s, t_e]$ is contained into another temporal interval $\Delta' = [t'_s, t'_e]$, denoted $\Delta \sqsubseteq \Delta'$, if $t'_s \leq t_s$ and $t'_e \geq t_e$.

Given an interval $\Delta \sqsubseteq T$, we denote $E_\Delta = \cap_{t \in \Delta} E_t$ the edges existing in all timestamps of Δ. Given an interval $\Delta \sqsubseteq T$, we denote $G_\Delta = (V, E_\Delta)$ as the static graph with vertices V and edges E_Δ.

We define the temporal support of an edge e over the temporal interval Δ to be equal to the support on the graph G_Δ, denoted as $\sup_\Delta(e)$.

Definition 3.1 ((k, Δ)-truss). The (k, Δ)-truss or *span-truss* of a temporal graph $G = (V, T, \tau)$ is the largest subgraph of G_Δ in which every edge is contained in at least ($k - 2$) triangles within the subgraph, i.e, $\sup_\Delta(e) \geq k - 2$, where $\Delta \sqsubseteq T$ is a temporal interval and $k \geq 2$. We will often denote the (k, Δ)-truss as $T_{k,\Delta}$.

A (k, Δ)-truss is a dense subgraph (where k is the cohesiveness constraint), together with its temporal span, i.e., the span Δ for which the subgraph satisfies the cohesiveness constraint.

Problem 3.1 (Span-truss decomposition). Given a temporal graph G, find the set of all (k, Δ)-trusses of G.

Observation 3.1. For a fixed temporal interval $\Delta \sqsubseteq T$, finding all span-trusses that have Δ as their span is equivalent to computing the classic truss decomposition of the static graph $G_\Delta = (V, E_\Delta)$.

Similarly to what has been proved for the span-cores [7], the total number of span-trusses may be too large for human inspection. In fact, the total number of temporal intervals contained in the whole time domain T is $\frac{|T|(|T|+1)}{2}$, so the total number of span-trusses is $O(|T|^2 \times k_{\max})$, where $k_{\max}$ is the largest value of k for which a (k, Δ)-truss exists. For this reason, it is worthwhile to focus only on the most important trusses, the maximal ones, as defined next.

Definition 3.2 (Maximal span-truss). A span-truss $T_{k,\Delta}$ of a temporal graph G is said to be *maximal* if there does not exist any other span-truss $T_{k',\Delta'}$ of G such that $k \leq k'$ and $\Delta \sqsubseteq \Delta'$.

A span-truss is recognized as maximal if it is not dominated by another span-truss both on order k and the span Δ. In our temporal setting, the number of maximal span-trusses is $O(|T|^2)$, as, in the worst case, there may be one maximal span-truss for every temporal interval. However, similarly to the maximal span-cores, we expect the number of maximal span-trusses to be much smaller.

Problem 3.2 (Maximal span-truss mining). Given a temporal graph G, find the set of all maximal (k, Δ)-trusses of G.

We now outline and prove some properties which will be useful later.

Proposition 3.1 (Span-truss containment). For any two span-trusses $T_{k,\Delta}, T_{k',\Delta'}$ of a temporal graph G, it holds that $k' \leq k \wedge \Delta' \sqsubseteq \Delta \implies T_{k,\Delta} \subseteq T_{k',\Delta'}$.

PROOF. The result can be proved by separately showing that (i) $k' \leq k \implies T_{k,\Delta} \subseteq T_{k',\Delta}$, and (ii) $\Delta' \sqsubseteq \Delta \implies T_{k,\Delta} \subseteq T_{k,\Delta'}$.
(i) holds because every $e \in E_\Delta$ is in at least k triangles in the subgraph $T_{k,\Delta}$, thus every e is also in at least k' triangles since $k' \leq k$; this means that $T_{k,\Delta} \subseteq T_{k',\Delta}$.
(ii) holds because $\Delta' \sqsubseteq \Delta \implies E_\Delta \subseteq E_{\Delta'} \implies \forall e \in E_\Delta, e \in E_{\Delta'}$. If e is in at least k triangles in $T_{k,\Delta}$ then it is in at least k triangles also in $T_{k,\Delta'}$, so $T_{k,\Delta} \subseteq T_{k,\Delta'}$. □

Definition 3.3 (Innermost truss). Let $T_{k^*}[G]$ denote the innermost truss of G, i.e., the non-empty k-truss of G with the largest k.
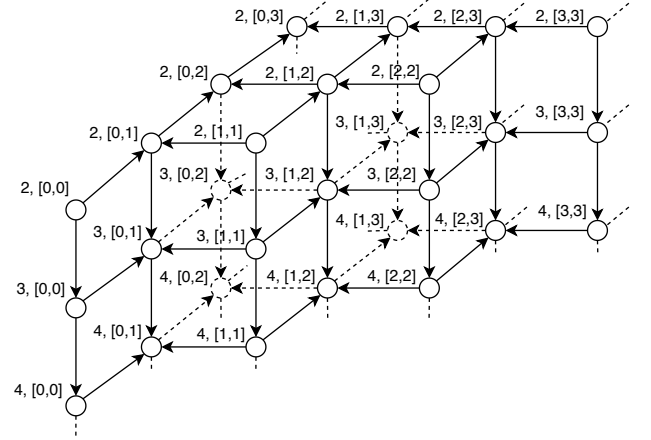

Figure 3: Graphical representation of the containment property. Span-trusses follow the same structure of span-cores. For a temporal span $\Delta = [t_s, t_e]$, the (k, Δ)-truss is depicted as a node labeled "$k, [t_s, t_e]$", an arrow $T_1 \rightarrow T_2$ denotes $T_1 \supseteq T_2$ [7].

Lemma 3.1. Given a temporal graph $G = (V, T, \tau)$, let T_M be the set of all maximal span-trusses of G, and $T_{inner} = \{T_{k^*}[G_\Delta] | \Delta \sqsubseteq T\}$ be the set of innermost trusses of all graphs G_Δ. It holds that $T_M \subseteq T_{inner}$.

PROOF. Every $T_{k,\Delta} \in T_M$ is the innermost truss of the non-temporal graph G_Δ: else, there would exist another truss $T_{k',\Delta} \neq \emptyset$ with $k' > k$, implying that $T_{k,\Delta} \notin T_M$. □

Lemma 3.2. Given a temporal graph $G = (V, T, \tau)$, and three temporal intervals $\Delta = [t_s, t_e] \sqsubseteq T$, $\Delta' = [t_s - 1, t_e] \sqsubseteq T$, and $\Delta'' = [t_s, t_e + 1] \sqsubseteq T$. The innermost truss $T_{k^*}[G_\Delta]$ is a maximal span-truss of G if and only if $k^* > \max\{k', k''\}$ where k' and k'' are the orders of the innermost trusses of $G_{\Delta'}$ and $G_{\Delta''}$, respectively.

PROOF. The "$\Rightarrow$" part comes directly from the definition of maximal span-truss (Definition 3.2): if k^* were not larger than $\max\{k', k''\}$, then $T_{k^*}[G_\Delta]$ would be dominated by another span-truss both on the order and on the span (as both Δ' and Δ'' are super intervals of Δ). For the "$\Leftarrow$" part, from Lemma 3.1 and Proposition 3.1 it follows that $\max\{k', k''\}$ is an upper bound on the maximum order of a span-truss of a super interval of Δ. Therefore, $k^* > \max\{k', k''\}$ implies that there cannot exist any other span-truss that dominates $T_{k^*}[G_\Delta]$ both on the order and on the span. □

4 EFFICIENT COMPUTATION OF MAXIMAL SPAN-TRUSSES

We present our solution by first giving a naïve approach, and then by introducing three versions (Baseline, Streaming, Heuristic) that improve over the previous version.

4.1 A naïve approach

A first approach to solve the problem could be based on Observation 3.1; namely, we can repeat the truss decomposition for every possible interval and then filter out non-maximal span-trusses.

Algorithm 1 Naïve maximal span-trusses

Input: A temporal graph $G = (V, T, \tau)$.
Output: The set T_M of all maximal span-trusses of G.

1: $candidates \leftarrow \emptyset$
2: $T_M \leftarrow \emptyset$
3: **forall** t_s in $[0, 1, \ldots, t_{max}]$ **do**
4:     **forall** t_e in $[t_s, t_s + 1, \ldots, t_{max}*] \mid E[t_s, t_e] \neq \emptyset$ **do**
5:         $\Delta \leftarrow [t_s, t_e]$
6:         $candidates[\Delta] \leftarrow computeMaxTruss(G_\Delta)$
7: $T_M \leftarrow$ maximal span-trusses from $candidates$

Algorithm 1 is trivially sound and complete since it iterates over every possible interval Δ, extracts the maximal k-truss from G_Δ and saves it as a candidate element of T_M.

T_M is constructed by filtering out non-maximal elements from $candidates$ and applying Definition 3.2.

4.2 Baseline Algorithm

As a baseline, we use a slightly better algorithm. This approach is similar to the baseline of the algorithm to mine span-cores [7]. It exploits the containment properties we have proved before, which are shared between span-cores and span-trusses.

Algorithm 2 Maximal span-trusses

Input: A temporal graph $G = (V, T, \tau)$.
Output: The set T_M of all maximal span-trusses of G.

1: $T_M \leftarrow \emptyset$
2: $K'[t] \leftarrow 0, \forall t \in T$
3: **forall** t_s in $[0, 1, \ldots, t_{max}]$ **do**
4:     $t^* \leftarrow \max\{t_e \in [t_s, t_{max}] \mid E[t_s, t_e] \neq \emptyset\}$
5:     $k'' \leftarrow 0$
6:     **forall** t_e in $[t^*, t^* - 1, \ldots, t_s]$ **do**
7:         $\Delta \leftarrow [t_s, t_e]$
8:         $lb \leftarrow \max\{K'[t_e], k''\}$
9:         $innermostTruss \leftarrow computeMaxTruss(G_\Delta)$
10:       $k^* \leftarrow$ order of $innermostTruss$
11:       **if** $k^* > lb$ **then**
12:           $T_M \leftarrow T_M \cup \{T\}$
13:       $k'' \leftarrow k^*$
14:       $K'[t_e] \leftarrow \max\{K'[t_e], k''\}$

Algorithm 2 works as follows. It iterates over all the starting timestamps $t_s \in T$ in increasing order and, for each t_s, all the maximal span-trusses that have span starting in t_s are identified. Proceeding in this way guarantees that a span-truss recognized as maximal will not be later dominated by another span-truss, since an interval $[t_s, t_e]$ can not be contained in another interval $[t'_s, t'_e]$ with $t_s < t'_s$.

To find all the maximal span-trusses having span starting in t_s, for any t_s the algorithm identifies $t^* \geq t_s$, the maximum timestamp

such that the edge set $E_{[t_s, t_e]}$ is not empty. Then, proceeding in decreasing order of t_e and starting from $t_e = t^*$, all intervals $\Delta = [t_s, t_e]$ are considered (from the largest interval to the smallest interval).

The internal cycle computes the lower bound lb (maximum between $K'[t_e]$ and k'') on the order of the innermost truss of G_Δ to be recognized as maximal. K' is a map that maintains, for every timestamp $t \in [t_s, t^*]$, the order of the innermost truss of graph G'_Δ where $\Delta = [t_s - 1, t]$ (i.e., $K'[t]$ stores what in Lemma 3.2 is denoted as k'). k'' stores the order of the innermost truss of G''_Δ and $\Delta'' = [t_s, t_e + 1]$.

The selected truss is added to the set of the maximal span-trusses only if its order is larger than lb, then the values of k'' and $K'[t_e]$ are updated.

Observation 4.1. The worst-case time complexity of Algorithm 2 is $O(|T|^2 \times |E|^{1.5})$ since the k-truss decomposition (complexity $O(|E|^{1.5})$) is repeated for every Δ. It is trivial to show that the number of possible intervals Δ is $O(|T|^2)$. Note that, since the output itself is potentially quadratic in $|T|$, it is not possible to improve over the $|T|^2$ factor in the computational complexity.

We outline now and discuss the operation of building the graph (V, E_Δ) efficiently on both space and time; we follow the approach of [7].

Having a fixed timestamp $t_s \in [0, \ldots, t_{max}]$, they propose the following reasoning which holds for every t_s. Let $E^-(t_e) = E_{[t_s, t_e]} \setminus E_{[t_s, t_e+1]}$ be the set of edges that are in $E_{[t_s, t_e]}$ but not in $E_{[t_s, t_e+1]}$, for $t_e \in [t_s, \ldots, t^*-1]$. For each t_s, one can compute and store all edge sets $\{E^-(t_e)\}_{t_e \in [t_s, t^*-1]}$. These operations can be done in $O(|T| \times |E|)$ time, because every $E^-(t_e)$ can be computed incrementally from $E_{[t_e, t_e]}$ as $E^-(t_e) = \{(u, v) \in E_{[t_s, t_e]} \mid \tau(u, v, t_e + 1) = 0\}$.

For any t_e, $E_{[t_s, t_e]}$ can be reconstructed as $E_{[t_s, t_e+1]} \cup E^-(t_e)$, having previously computed $E_{[t_s, t_e+1]}$. Note that storing all $E^-(t_e)$ takes $O(|E|)$ space. That is why all $E^-(t_e)$ are stored and $E_{[t_s, t_e]}$ are reconstructed afterward instead of storing the latter, which would take $O(|T| \times |E|)$ space.

We use this approach in Algorithm 2.

Observation 4.2. Since for any t_e, we reconstruct $E_{[t_s, t_e]}$ as $E_{[t_s, t_e+1]} \cup E^-(t_e)$, we are always adding new edges to the graph $G_{[t_s, t_e+1]}$ starting from an empty graph. This means we can exploit a streaming approach to solve the problem.

4.3 A Streaming Algorithm

It is trivial to see that the Algorithm 2 repeats the truss decomposition in every possible interval. This means it also repeats the support computation, which for a single interval Δ has complexity $O(|E_\Delta|^{1.5})$ and it is the most expensive operation. Here we outline an algorithm to achieve better performance with regards to the support computation.

We can reframe the problem and think of it as a streaming problem, as stated in Observation 4.2. Suppose we have computed the support for every edge active in the interval $\Delta^* = [t_s, t_e + 1]$. In the next step, we consider the interval $\Delta = [t_s, t_e]$ and so we are considering the graph G_Δ which is simply the graph G_{Δ^*} with a number of edges added, namely $E^-(t_e)$. We can study how the addition of these new edges changes the support of the edges of
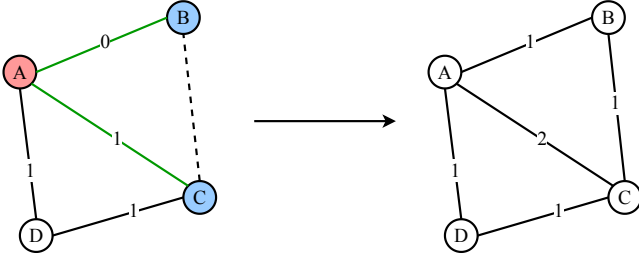
Figure 4: In this example we show how the insertion of a new edge (B, C) affects the supports of the other edges in the graph. The red vertex A is the only vertex in $(\textbf{neighbours}(B) \cap \textbf{neighbours}(C))$, so we update the supports of (A, C), (A, B) and of the new edge (B, C). In fact we observe that (B, C) forms a triangle with these edges, colored in green. On the right, we have the graph with the supports updated.

the old graph G_{Δ^*} and develop an algorithm that computes only the support of the edges in $E^-(t_e)$ and just *updates* the support of the edges in G_{Δ^*}. The updating part, without always recomputing, leads to a high speedup in the performance, as we will see in the next section.

After the update of the support of the edges, we can run the truss decomposition algorithm.

Algorithm 3 Computing the support of every edge in G_Δ efficiently

---
Input: A graph $G_{[t_s, t_e+1]} = (V, E_{[t_s, t_e+1]})$ with the support computed for every edge and a set $E^-(t_e)$ of edges to add to
$$G_{[t_s, t_e+1]}$$
Output: A graph $G_{[t_s, t_e]} = (V, E_{[t_s, t_e+1]} \cup E^-(t_e))$ with the supports updated

1: **forall** $e \in E^-(t_e)$ **do**
2:      add e to $G_{[t_s, t_e+1]}$
3:      let $(u, v) = e$
4:      **forall** $w \in (\text{neighbours}(u) \cap \text{neighbours}(v))$ **do**
5:          $\sup(u, v) = \sup(u, v) + 1$
6:          $\sup(v, w) = \sup(v, w) + 1$
7:          $\sup(u, w) = \sup(u, w) + 1$

Observation 4.3. If we use a map M, which maps a pair of vertices (u, v) to 1 if the edge exists in G_Δ at observation time or to 0 if it does not exists, we can implement the intersection at step 4 by simply iterating over the neighbours of one of the two vertices and check in $O(1)$ if the remaining edge to form the triangle exists in the graph at observation time. Hence, the running time of this approach is bounded by $\sum_{(u,v) \in E^-(t_e)} \min\{\deg(u), \deg(v)\}$.

4.4 Applying heuristics

It is worth mentioning that we still compute the truss decomposition in every graph G_Δ. From Algorithm 2, lines 11 to 14, we observe that a k-truss recognized as a maximal k-truss in a snapshot of a temporal graph will not always be recognized as a maximal span-truss.

Observation 4.4. If the order of the innermost-truss I' of the graph $G_{[t_s, t_e]}$ is k' and the order of the innermost-truss I'' of the graph $G_{[t_s, t_{e-1}]}$ is k' then I'' is not a maximal span-truss.

Observation 4.5. If the order of the innermost-truss I' of the graph $G_{[t_s, t_e]}$ is k' and the graph $G_{[t_s, t_{e-1}]}$ and the graph $G_{[t_s, t_e]}$ have the same number of edges with support greater than $k' - 2$ then the order of I'' is k'.

These two simple yet effective observations provide a minimal condition to avoid the computation of the truss decomposition in a snapshot of a temporal graph and lead to an improvement in the performance in particular datasets, as we will see in the next chapter.

5 EVALUATION

Datasets. We use eight real-world datasets recording timestamped interactions between entities[1], as in [7]. For each dataset, a window size is selected to build the corresponding temporal graph. Multiple interactions occurrinng between two entities during the same discrete timestamp are counted as one. The characteristics of the resulting graphs are reported in Table 1.

PROSPERLOANS represents the network of loans between the users of Prosper, a marketplace of loans between privates. LASTFM records the co-listening activity of the streaming platform Last.fm: two users are connected if they listened to songs of the same band during the same discrete timestamp. WIKITALK is the communication network of the English Wikipedia. DBLP is the co-authorship network of the authors of scientific papers from the DBLP computer science bibliography. STACKOVERFLOW includes the answer-to-question interactions on StackOverflow. WIKIPEDIA connects users of the Italian Wikipedia that co-edited a page within the same discrete timestamp. In the AMAZON dataset, vertices are users, and edges represent the rating of at least one common item within the same discrete timestamp.

Implementation. The code[2] for the experiments has been implemented in C++11, compiled with g++ 5.4 and -O3 optimization, and run on a machine equipped with a 2,2 GHz CPU, 94GB RAM and Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-145-generic x86_64).

---

[1]All datasets are made available by the KONECT Project (http://konect.cc), except for StackOverflow which is part of the SNAP Repository (http://snap.stanford.edu).
[2]https://github.com/FraLotito/span_trusses

Dataset	$|V|$	$|E|$	$|T|$	window size (days)	domain
PROSPERLOANS	89k	3M	307	7	economic
LASTFM	992	4M	77	21	co-listening
WIKITALK	2M	10M	192	28	communication
DBLP	1M	11M	80	366	co-authorship
STACKOVERFLOW	2M	16M	51	56	question answering
WIKIPEDIA	343k	18M	101	56	co-editing
AMAZON	2M	22M	115	28	co-rating

Table 1: Description of the temporal graphs used for the experiments

Dataset	# maximal span-trusses
PROSPERLOANS	293
LASTFM	1539
WIKITALK	466
DBLP	268
STACKOVERFLOW	112
WIKIPEDIA	1905
AMAZON	303

Table 2: Number of maximal span-trusses in each dataset

Dataset	Baseline (s)	Streaming (s)	Heuristics (s)
PROSPERLOANS	5	5	5
LASTFM	1318	**1057**	1109
WIKITALK	7497	818	**336**
DBLP	513	112	**85**
STACKOVERFLOW	381	91	**63**
WIKIPEDIA	2447	**1731**	1837
AMAZON	3025	**2598**	2607

Table 3: Experimental results

Results. Table 2 reports the number of maximal span-trusses that are present in the datasets.

Table 3, instead, shows the computing time for each of the datasets for the Baseline, Streaming and Heuristic algorithms. The table shows how computing the support of the edges in a streaming fashion improves the overall performance of the algorithm. We report a constant decrease in the time execution, with a peak with the WIKITALK dataset, which takes almost ten times less than the baseline.

The table also shows how our proposed heuristic to avoid unnecessary decompositions helps in reducing the time execution in some of the datasets, with a peak with the WIKITALK dataset which takes half the time with respect to our efficient algorithm. In some datasets, however, the heuristic comes with minimal overhead; we believe that it is worthwhile to use such version anyway, to exploit the more significant performance gain in the other cases.

6 RELATED WORK

The first and most obvious dense subgraph introduced to social network analysis is the clique, a subgraph in which every vertex is adjacent to every other vertex [13]. Computing cliques has several disadvantages. First, they are both too rare and too common: cliques of only a few members are frequently too numerous to be helpful, while larger cliques are too difficult to be found in real-world graphs. Second, no polynomial-time algorithm is known for this problem: this makes the enumeration of cliques impractical for moderate data sizes [3].

A number of generalizations and relaxations have been proposed to avoid the issues of rarity and tractability of cliques [1, 15, 18].

A well-known relaxation of the clique is the k-core decomposition [17]. A k-core is a maximal subgraph in which each member is adjacent to at least k other members. Unlike other clique generalizations, k-cores can be computed and listed in polynomial time.
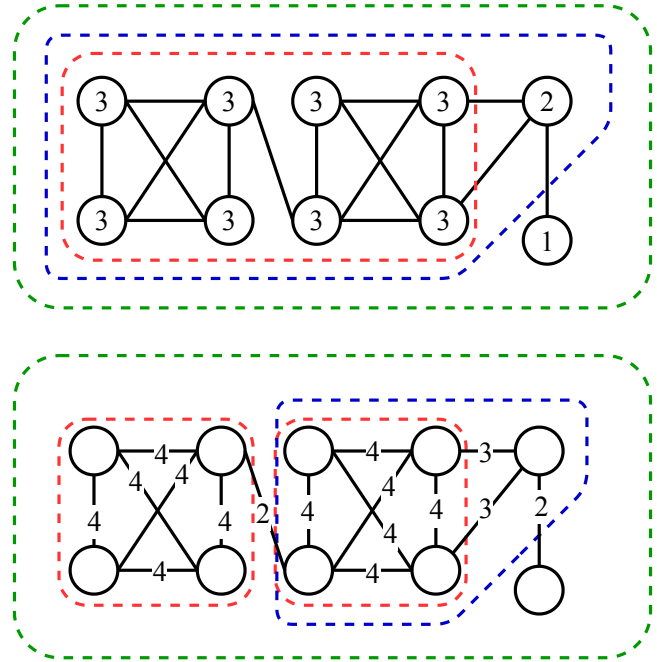

Figure 5: Example of the differences between k-core (first picture) and k-truss decomposition (second picture) [12]. We highlight the coreness of every vertex in the first picture and the trussness of every edge in the second.

The disadvantage of k-cores is that they are too promiscuous and they can be of questionable utility.

The concept k-truss has been introduced as a compromise between the expensive-to-find and overly-numerous groupings provided by cliques, k-cliques, k-clubs, k-plexes on the one hand, and the easy-to-compute, few-in-number, but overly-generous k-cores on the other [6]. In most real-world graphs, the maximum trussness is much lower than the maximum coreness, and the highest order truss is much denser than the highest-order core [20]. Figure 5 highlights the differences between k-core and k-truss.

Recently, there has been an increasing interest from the research community in generalizing cohesive structure concepts in a temporal setting. Our work is directly inspired by the work of Galimberti et al. [7] who generalized the concept of k-core and introduced the concept of *span-core*. They also provided the corresponding algorithms to compute all the span-cores and to efficiently compute only the maximal ones (span-cores that are not dominated by any other span-core by both the coreness property and the span) in a temporal graph.

Other works related to ours include Semertzidis et al. [19], who introduced the problem of identifying a set of vertices that are densely connected in at least k timestamps of a temporal network; Himmel at al. [8] and Viard et al. [21], who generalized the concept of clique in a temporal graph and proposed the respective listing algorithms; and Ma et al. [14], who a proposed a statistics-driven approach to find dense temporal subgraphs in large temporal networks.

7 CONCLUSIONS

In this paper, we have generalized the concept of k-truss to a temporal setting defining a structure called *span-truss*, where each truss is associated with its span. We have developed both a naïve and an efficient algorithm to extract all the maximal span-trusses of a temporal graph, along with a heuristic to improve the running time in particular conditions. Finally, we have evaluated our proposals on a number of public datasets.

In our future work, we plan to explore new heuristics to avoid the computation of the whole truss decomposition when not needed; for example, Burkhardt et al. [4] summarized a number of properties and bounds that a k-truss must satisfy and which can be useful to avoid the computation of the decomposition when not needed.

REFERENCES

[1] Richard D. Alba. 1973. A Graph-Theoretic Definition of a Sociometric Clique. *Journal of Mathematical Sociology* 3 (1973), 3–113. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.8697

[2] Albert Angel, Nikos Sarkas, Nick Koudas, and Divesh Srivastava. 2012. Dense Subgraph Maintenance Under Streaming Edge Weight Updates for Real-time Story Identification. *Proc. VLDB Endow.* 5, 6 (2012), 574–585. https://doi.org/10.14778/2168651.2168658

[3] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: Finding All Cliques of an Undirected Graph. *Commun. ACM* 16, 9 (Sept. 1973), 575–577. https://doi.org/10.1145/362342.362367

[4] Paul Burkhardt, Vance Faber, and David G. Harris. 2018. Bounds and algorithms for k-truss. arXiv:math.CO/1806.05523

[5] Lijun Chang. 2018. *Cohesive subgraph computation over large sparse graphs : algorithms, data structures, and programming techniques.* Springer, Boston, MA.

[6] Jonathan Cohen. 2008. Trusses: Cohesive Subgraphs for Social Network Analysis.

[7] Edoardo Galimberti, Alain Barrat, Francesco Bonchi, Ciro Cattuto, and Francesco Gullo. 2018. Mining (Maximal) Span-Cores from Temporal Networks. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (Torino, Italy) *(CIKM '18).* Association for Computing Machinery, New York, NY, USA, 107–116. https://doi.org/10.1145/3269206.3271767

[8] A. Himmel, H. Molter, R. Niedermeier, and M. Sorge. 2016. Enumerating maximal cliques in temporal graphs. In *IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM'16).* IEEE Press, Calgary, Canada, 337–344. https://doi.org/10.1109/ASONAM.2016.7752255

[9] Petter Holme and Jari Saramäki. 2012. Temporal networks. *Physics Reports* 519, 3 (Oct 2012), 97–125. https://doi.org/10.1016/j.physrep.2012.03.001

[10] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying K-Truss Community in Large and Dynamic Graphs. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14).* Association for Computing Machinery, New York, NY, USA, 1311–1322. https://doi.org/10.1145/2588555.2610495

[11] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu C. Aggarwal. 2010. A Survey of Algorithms for Dense Subgraph Discovery. In *Managing and Mining Graph Data.* Springer, Boston, MA, 303–336.

[12] Penghang Liu and A. Erdem Sarıyüce. 2019. Analysis of Core and Truss Decompositions on Real-World Networks. In *Proceedings of the 15th International Workshop on Mining and Learning with Graphs (MLG).*

[13] R. Duncan Luce and Albert D. Perry. 1949. A method of matrix analysis of group structure. *Psychometrika* 14, 2 (01 Jun 1949), 95–116. https://doi.org/10.1007/BF02289146

[14] Shuai Ma, Renjun Hu, Luoshu Wang, Xuelian Lin, and Jin-Peng Huai. 2019. An Efficient Approach to Finding Dense Temporal Subgraphs. *IEEE Transactions on Knowledge and Data Engineering* PP (01 2019), 1–1. https://doi.org/10.1109/TKDE.2019.2891604

[15] Robert J. Mokken. 1979. Cliques, clubs and clans. *Quality and Quantity* 13, 2 (01 Apr 1979), 161–173. https://doi.org/10.1007/BF00139635

[16] Thomas Schank. 2007. *Algorithmic Aspects of Triangle-Based Network Analysis.* Ph.D. Dissertation. Universität Karlsruhe, Karlsruhe. https://doi.org/10.5445/IR/1000007159

[17] Stephen B. Seidman. 1983. Network structure and minimum degree. *Social Networks* 5, 3 (1983), 269 – 287. https://doi.org/10.1016/0378-8733(83)90028-X

[18] Stephen B. Seidman and Brian L. Foster. 1978. A graph-theoretic generalization of the clique concept. *The Journal of Mathematical Sociology* 6, 1 (1978), 139–154. https://doi.org/10.1080/0022250X.1978.9989883

[19] Konstantinos Semertzidis, Evaggelia Pitoura, Evimaria Terzi, and Panayiotis Tsaparas. 2018. Finding lasting dense subgraphs. *Data Mining and Knowledge Discovery* 33, 5 (Nov. 2018), 1417–1445. https://doi.org/10.1007/s10618-018-0602-x

[20] Kijung Shin, Tina Eliassi-Rad, and Christos Faloutsos. 2018. Patterns and Anomalies in K-Cores of Real-World Graphs with Applications. *Knowl. Inf. Syst.* 54, 3 (March 2018), 677–710. https://doi.org/10.1007/s10115-017-1077-6

[21] Jordan Viard, Matthieu Latapy, and Clémence Magnien. 2015. Revealing Contact Patterns among High-School Students Using Maximal Cliques in Link Streams. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015* (Paris, France) *(ASONAM '15).* Association for Computing Machinery, New York, NY, USA, 1517–1522. https://doi.org/10.1145/2808797.2809291

[22] Jia Wang and James Cheng. 2012. Truss Decomposition in Massive Networks. *PVLDB* 5, 9 (2012), 812–823. https://doi.org/10.14778/2311906.2311909