Dense Subgraph Discovery Meets Strong Triadic Closure

Chamalee Wickrama Arachchi
University of Helsinki
Helsinki, Finland
chamalee.wickramaarachch@helsinki.fi

Iiro Kumpulainen
University of Helsinki
Helsinki, Finland
iiro.kumpulainen@helsinki.fi

Nikolaj Tatti
HIIT, University of Helsinki
Helsinki, Finland
nikolaj.tatti@helsinki.fi

Abstract

Finding dense subgraphs is a core problem with numerous graph mining applications such as community detection in social networks and anomaly detection. However, in many real-world networks connections are not equal. One way to label edges as either strong or weak is to use strong triadic closure (STC). Here, if one node connects strongly with two other nodes, then those two nodes should be connected at least with a weak edge. STC-labelings are not unique and finding the maximum number of strong edges is **NP**-hard. In this paper, we apply STC to dense subgraph discovery. More formally, our score for a given subgraph is the ratio between the sum of the number of strong edges and weak edges, weighted by a user parameter λ, and the number of nodes of the subgraph. Our goal is to find a subgraph and an STC-labeling maximizing the score. We show that for $\lambda = 1$, our problem is equivalent to finding the densest subgraph, while for $\lambda = 0$, our problem is equivalent to finding the largest clique, making our problem **NP**-hard. We propose an exact algorithm based on integer linear programming and four practical polynomial-time heuristics. We present an extensive experimental study that shows that our algorithms can find the ground truth in synthetic datasets and run efficiently in real-world datasets.

CCS Concepts

• **Theory of computation → Graph algorithms analysis**.

Keywords

dense subgraph, strong triadic closure, integer linear programming

ACM Reference Format:
Chamalee Wickrama Arachchi, Iiro Kumpulainen, and Nikolaj Tatti. 2024. Dense Subgraph Discovery Meets Strong Triadic Closure. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3637528.3671697

1 Introduction

Many social networks naturally contain both *strongly* connected and *weakly* connected interactions among the entities of the network. A question of particular interest is that given a set of pairwise user interactions, how to infer the strength of the social ties within the network? In other words, how to label the edges of an undirected graph as either strong or weak?

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671697

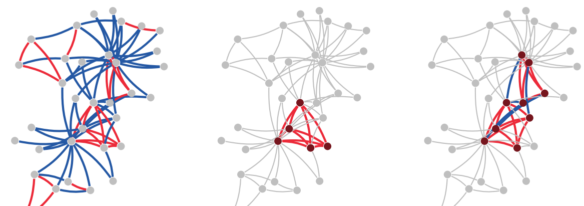

Figure 1: Strong (Red) and weak (Blue) edges of the Karate club dataset maximizing the number of strong edges (a), $\lambda = 0$ (b), and $\lambda = 0.5$ (c) using our integer linear program based algorithm (STC-ILP). We define our score as the sum of the number of strong and weak edges weighted by a parameter λ, divided by the size of the subgraph. The scores are 2.0 and 2.06 for (b) and (c), respectively. We see that (b) is a clique of size 5.

The problem of inferring the strength of social ties based on *strong triadic closure* principle (STC) has drawn attention over the past decade within the data mining community [1, 14, 16, 17, 21, 24]. The STC property assumes that there exist two types of social ties in the network: either *strong* or *weak*. Let A, B, and C be three entities in the network. If the entities A and B are strongly connected with the entity C, then there should be at least a weak connection between A and B. In other words, if both A and B are strong friends of C, then some kind of connection between A and B should also exist. Note that these labels are not known and the goal is to infer the labels from the given unlabeled graph.

In this paper, we incorporate the STC property into the problem of dense subgraph discovery [9, 24]. More formally, given a subgraph and a weight parameter λ, we define a score as the ratio between the sum of the number of strong and weak edges weighted by λ and the number of nodes within the subgraph. Our objective is to find a subgraph *and* a labeling that maximize our score while satisfying the STC property within the subgraph.

We will see that when $\lambda = 0$ finding an optimal subgraph is equal to finding a maximum clique. On the other hand, for $\lambda = 1$, finding an optimal subgraph is equal to finding the densest subgraph, that is, a subgraph U maximizing the ratio of edges and nodes, $|E(U)|/|U|$. Both of these problems are well-studied. Optimizing the score for $0 < \lambda < 1$, yields a problem that is between these two cases. We expect that for small λs the returned subgraph resembles a clique whereas large λs yield a subgraph similar to the densest subgraph.

Example: We illustrate the difference between our problem and the original STC problem considered by Sintos and Tsaparas [24] in Figure 1. The goal of this paper is to find a subgraph that maximizes our score while satisfying the STC property. In contrast, Sintos and

Tsaparas [24] aims to label *all* the edges in the graph such that the number of weak edges is minimized. Figure 1 shows the results obtained with the Karate club dataset with our exact algorithm. We should stress that the labeling of the discovered *subgraph* might be different from the labeling that maximizes strong edges for the whole graph. In Figure 1 (c), we see that 4 weak edges have been turned into strong while the labeling of the remaining edges is unchanged.

We show that our problem is **NP**-hard when $\lambda < 1$, and even inapproximable when $\lambda = 0$ since our problem then reduces to the Max-Clique problem. However, the $\lambda = 1$ case is solvable in polynomial time. To solve the problem, we first propose an exact algorithm based on integer linear programming which runs in exponential time. We consider four other heuristics that run in polynomial time in the size of the input graph: We propose a linear linear programming based heuristic in Section 5.2 and a greedy algorithm in Section 5.4. We also propose two straightforward algorithms that combine the existing algorithms for solving the densest subgraph problem and finding STC-compliant labeling in an entire graph.

The remainder of the paper is organized as follows. First, we introduce preliminary notation and our problem in Section 2. Next, we present the related work in Section 3. Next, we show that our problem is **NP**-hard in Section 4 and then explain our algorithms in Section 5. Finally, we present our experimental evaluation in Section 6 and conclude the paper with a discussion in Section 7.

2 Preliminary notation and problem definition

We begin by providing preliminary notation and formally defining our problem.

Our input is an unweighted graph $G = (V, E)$, and we denote the number of nodes and edges by $n = |V|$ and $m = |E|$. Given a graph $G = (V, E)$ and a set of nodes $U \subseteq V$, we define $E(U) \subseteq E$ to be the subset of edges having both endpoints in U. We denote the degree of vertex v as $\deg(v)$. We denote the set of adjacent edges of vertex v as $N(v)$.

We want to label the set of edges E as either *strong* or *weak*. To perform the labeling, we use the *strong triadic closure* (STC) property [24]. We say that the graph is STC satisfied if for any given triplet of vertices, (x, y, z) the following holds: if (x, y) and (y, z) are connected and are labeled as strongly connected, then the edge (x, z) always exists at least as a weak edge.

We call a triplet of vertices (x, y, z) a *wedge*, if $(x, y) \in E$, $(y, z) \in E$, and $(x, z) \notin E$. A *wedge graph* $Z(G)$ consists of the edges of the graph G that contribute to at least one wedge as its vertices. If the two edges e_1 and e_2 of G form a wedge, we add an edge between the two nodes in $Z(G)$ that corresponds to e_1 and e_2, that is, each edge of $Z(G)$ corresponds to a wedge of G.

Given a graph $G = (V, E)$ and a labeling L of the edges as strong or weak, we write $E_s(U, L)$ and $E_w(U, L)$ to be the set of strong and weak edges of the graph induced by a set of vertices U. We also write $m_s(U, L) = |E_s(U, L)|$ and $m_w(U, L) = |E_w(U, L)|$. Finally, for a vertex $u \in U$, we define a strong and weak degree, $\deg_s(u, U, L)$ and $\deg_w(u, U, L)$ to be the number of strong or weak edges in U adjacent to u. We may omit L or U from the notation when it is clear from the context.

Given a weight parameter λ where $\lambda \in [0, 1]$ and a label assignment L, we define a score

$$q(U, L; \lambda) = \frac{m_s(U, L) + \lambda m_w(U, L)}{|U|} \quad .$$

We may omit L or λ when it is clear from the context.

We consider the following optimization problem.

Problem 1 (stc-den). *Given a graph $G = (V, E)$ and a weight parameter λ, find a subset of vertices $U \subseteq V$ and a labeling L of the edges as strong or weak such that the STC property is satisfied in $(U, E(U))$ and $q(U, L; \lambda)$ is maximized.*

Note that when $\lambda = 1$ then the labeling does not matter, and stc-den reduces to dense subgraph discovery, that is, finding U with the largest ratio $|E(U)|/|U|$. On the other hand, if $\lambda = 0$, then we only take into account strong edges; we will show in Section 4 that in this case, stc-den is equal to finding a maximum clique.

Note also that the labeling depends on the underlying subgraph U, that is, we need to find U and L simultaneously.

3 Related work

The strong triadic closure (STC) property: As an early work in this line of research, Sintos and Tsaparas [24] considered the problem of minimizing the number of weak edges (analogously maximizing the number of strong edges) while labeling the edges compliant with the STC property. We will refer to this optimization problem as MinSTC.

Sintos and Tsaparas [24] showed that MinSTC is equivalent to solving a minimum vertex cover, which we denote by Min-Vertex-cover, in a wedge graph $Z(G)$. In Min-Vertex-cover, we search for a minimum number of nodes such that at least one endpoint of each edge is in the set.

Sintos and Tsaparas [24] proposed the following algorithm for MinSTC, which we denote by Apr-MinSTC. Given a graph G they first construct the wedge graph $Z(G)$ and find its vertex cover. Next, they label the set of edges of G that corresponds to the set of vertices in the vertex cover as weak, and the remaining edges as strong. Since Min-Vertex-cover is **NP**-hard, they approximate it with a maximal matching algorithm [6]. The algorithm picks an arbitrary edge and adds both endpoints of the edge to the cover, and the edges incident to both endpoints are deleted. It continues until no edges are left. This algorithm outputs a *maximal matching* which is known to yield a 2-approximation since at least one endpoint should be in the cover.

The problem of finding a labeling of edges that maximizes the number of strong edges while satisfying the STC property is **NP**-hard for general graphs [24] and split graphs [15]. Nevertheless, it becomes polynomial-time solvable for proper interval graphs [15], cographs [14], and trivially perfect graphs [15]. Given communities, Rozenshtein et al. [21] considered the problem of inferring strengths while minimizing STC violations with additional connectivity constraints. Oettershagen et al. [17] extended the idea of inferring tie strength for temporal networks. Matakos and Gionis [16] considered the problem of strengthening edges to maximize STC violations, which they consider as opportunities to build new connections. Adriaens et al. [1] formulated both minimization and maximization versions of STC problems as linear programs.

The dense subgraph discovery: Finding dense subgraphs is a core problem in social network analysis. Given a graph, the densest subgraph problem is defined as finding a subgraph with the highest average degree density (twice the number of edges divided by the number of nodes). Finding the densest subgraph for a single graph has been extensively studied [5, 9, 13]. Goldberg [9] proposed an exact, polynomial time algorithm that solves a sequence of min-cut instances. We will refer to this algorithm as CUT. Asahiro et al. [3] provided a greedy algorithm and Charikar [5] proved that their greedy algorithm gives a 1/2-approximation, showed how to implement the algorithm using Fibonacci heaps, and devised a linear-programming formulation of the problem. The idea of the greedy algorithm is that at each iteration, a vertex with the minimum degree is removed, and then the densest subgraph among all the produced subgraphs is returned as the solution. We will refer to this algorithm as PEEL. It has also been extended for multiple graph snapshots [2, 11, 23]. The problem has been studied in a streaming setting [4]. To the best of our knowledge, this is the first attempt to consider the notion of density together with the STC property.

In addition to degree density (a.k.a average degree), alternative types of density measures have also been considered previously such as triangle density and k-clique density [26]. Triangle density is defined as the ratio between the number of triangles and the number of vertices of the subgraph. The definition that will be used in this paper is the ratio between the number of induced edges and nodes. Adopting our problem to other density measures is left open as future work.

4 Computational complexity

In this section, we analyze the computational complexity of the STC-DEN problem by showing its **NP**-hardness when $\lambda < 1$ and its connection to the MAX-CLIQUE problem, when $\lambda = 0$. At the other extreme, when $\lambda = 1$, STC-DEN is equivalent to the problem of finding the densest subgraph, which can be solved in polynomial time using the algorithm presented by Goldberg [9].

PROPOSITION 4.1. *For $\lambda = 0$, STC-DEN is **NP**-hard.*

The proof (given in Appendix A.1) also shows that the maximum clique yields precisely the optimal score while satisfying the STC property. As a consequence, combining this with the inapproximability results for MAX-CLIQUE [28] gives the next result (see Appendix A.1 for the proof).

PROPOSITION 4.2. *For $\lambda = 0$, STC-DEN does not have any polynomial time approximation algorithm with an approximation ratio better than $n^{1-\epsilon}$ for any constant $\epsilon > 0$, unless **P** = **NP**.*

Note that when $\lambda > 0$, we can obtain a $\frac{1}{\lambda}$-approximation by finding the densest subgraph with nodes U and using a labeling L that labels each edge as weak. Compared to an optimal solution U^* with labeling L^*, we then obtain the score

$$\begin{aligned} q(U, L; \lambda) = \frac{\lambda |E(U)|}{|U|} &\geq \frac{\lambda |E(U^*)|}{|U^*|} = \lambda \frac{m_s(U^*, L^*) + m_w(U^*, L^*)}{|U^*|} \\ &\geq \lambda \frac{m_s(U^*, L^*) + \lambda m_w(U^*, L^*)}{|U^*|} = \lambda q(U^*, L^*; \lambda). \end{aligned}$$

In summary, STC-DEN is inapproximable when $\lambda = 0$ but solvable in polynomial time when $\lambda = 1$. Finally, we state that STC-DEN is also **NP**-hard for $0 < \lambda < 1$.

PROPOSITION 4.3. *STC-DEN is **NP**-hard for $0 < \lambda < 1$.*

The proof of Proposition 4.3 is in Appendix A.1.

5 Algorithms

In this section, we present five algorithms to find a good subgraph for our STC-DEN problem. First, we propose an algorithm based on integer linear programming that finds a near-optimal or exact solution for our problem in Section 5.1. Next, we state a polynomial time algorithm that solves a linear program in Section 5.2 followed by three heuristics presented in Sections 5.3 and 5.4.

5.1 Exact solution using integer programming

In this section, we present an integer linear programming (ILP) based algorithm that can be used to find an exact solution for STC-DEN. To solve STC-DEN we need the following auxiliary problem. The proofs for this section are given in Appendix A.2.

PROBLEM 2 (STC-DEN(α)). *Given a graph $G = (V, E)$, a weight parameter λ, and a number α, find a subset of vertices $U \subseteq V$ and a labeling L of the edges such that the STC property is satisfied in $(U, E(U))$ and $m_s(U, L) + \lambda m_w(U, L) - \alpha|U|$ is maximized.*

The following proposition, which is an instance of fractional programming [8], shows the relationship between STC-DEN(α) and STC-DEN.

PROPOSITION 5.1. *Let $U(\alpha)$ and $L(\alpha)$ be the subgraph and the corresponding labeling solving STC-DEN(α). Similarly, let U^* with labeling L^* be the solution for STC-DEN. Write $\alpha^* = q(U^*, L^*)$. If $\alpha > \alpha^*$, then $U(\alpha) = \emptyset$. If $\alpha < \alpha^*$, then $U(\alpha) \neq \emptyset$ and $q(U(\alpha), L(\alpha)) > \alpha$.*

We can use the proposition to solve STC-DEN: we find the (almost) largest α for which STC-DEN(α) yields a nonempty solution. Then STC-DEN(α) for such α yields an (almost) optimal solution.

We can solve STC-DEN(α) with an integer linear program,

$$\begin{array}{llrr} \text{maximize} & \sum x_{ij} + \lambda \sum z_{ij} - \alpha \sum y_i & & (1) \\ \text{subject to} & x_{ij} + z_{ij} \leq y_i & ij \in E & (2) \\ & x_{ij} + z_{ij} \leq y_j & ij \in E & (3) \\ & x_{ij} + x_{jk} \leq y_j & (i, j, k) \in Z & (4) \\ & x_{ij}, z_{ij} \in \{0, 1\} & ij \in E & (5) \\ & y_i \in \{0, 1\} & i \in V\ . & (6) \end{array}$$

Here, $G = (V, E)$ is the input graph and Z is the set of all wedges in G.

To see why this program solves STC-DEN(α), let $S \subseteq V$ and L be a solution to our STC-DEN(α). The indicator variable y_i denotes whether the node $i \in S$ or not. The indicator variables x_{ij} and z_{ij} denote if the edge (i, j) is strong or not and (i, j) is weak or not, respectively. Constraints 2-3 guarantee that each edge within S is labeled either as strong or weak. Constraint 4 ensures that the STC constraint is satisfied.

Proposition 5.1 allows us to maximize α with a binary search. Here, we set the initial interval (L, U) to $L = 0$ and $U = \frac{n-1}{2}$, and keep halving the interval until $|U - L| \leq \epsilon L$, where $\epsilon > 0$ is an input parameter, and return the solution to STC-DEN(L). We refer to this algorithm as STC-ILP. Next, we state that STC-ILP yields an approximation guarantee of $1/(1 + \epsilon)$.

Proposition 5.2. *Assume a graph $G = (V, E)$, $\lambda \in [0,\ 1]$, and $\epsilon > 0$. Let α be the score of the solution returned by STC-ILP and let α^* be the optimal score of STC-DEN. Then $\alpha \geq \alpha^*/(1 + \epsilon)$.*

Next, we will show that if ϵ is small enough, we are guaranteed to find the exact solution.

Proposition 5.3. *Assume a graph G with n nodes. Assume that the weight parameter λ is a rational number $\lambda = \frac{a}{b}$. Then, if we set $\epsilon = \frac{2}{bn^3}$, STC-ILP returns an exact solution for the STC-DEN problem in $O(\log n + \log b)$ number of iterations.*

STC-ILP requires $O(\log n - \log \epsilon)$ iterations, solving an integer linear program in each round. Note that solving an ILP is **NP**-hard [22], and the fastest known algorithm to solve an ILP exactly runs in $\log h^{O(h)}$ time where h is the number of variables [19]. In practice, we can solve STC-DEN(α) for moderately sized graphs but for larger graphs solving the ILP becomes computationally infeasible.

This approach is related to two prior works. First, the algorithm by Goldberg [9] for finding the densest subgraph problem uses a similar approach, except without the variables z_{ij}. In such a case, the program can be solved exactly with a minimum cut. Secondly, Adriaens et al. [1] use a linear program with similar wedge constraints to approximate MinSTC.

5.2 Algorithm based on linear programming

In this section, we present an algorithm, named STC-LP, based on a linear program obtained by relaxing the integrality requirements of the integer linear program given in the previous section. More specifically, we first find a fractional solution by solving a linear program (LP) and then derive a good subgraph via a rounding algorithm. Note that solving linear programs can be done in polynomial time [12, 27], and solvers are efficient in practice. The proofs for this section are given in Appendix A.3.

Consider a relaxed version of the ILP, where we replace the constraints in Eqs. 5–6 with $x_{ij}, z_{ij} \in [0, 1]$ and $y_i \in [0, 1]$. We will refer to this optimization problem as STC-RELAX(α).

Note that we used STC-DEN(α) combined with the binary search to solve STC-DEN. We can define a relaxed version of STC-DEN which then can be analogously solved with STC-RELAX(α).

Problem 3 (STC-RELAX). *Given a graph $G = (V, E)$, a weight parameter λ, find a nonnegative set of variables x_e, y_i, z_e, where $e \in E$ and $i \in V$ maximizing*

$$r(x, y, z) = \frac{\sum x_e + \lambda \sum z_e}{\sum y_i} \quad \textit{such that} \quad \textit{Eqs. 2–4 hold.}$$

STC-RELAX is a relaxed version of STC-DEN: if we were to require that the variables in STC-RELAX are binary numbers, then the problems become equivalent. The next proposition is an analog to Proposition 5.1.

Proposition 5.4. *Let (x^*, y^*, z^*) be a solution to STC-RELAX. Write $\alpha^* = r(x^*, y^*, z^*)$. Similarly, let $(x(\alpha), y(\alpha), z(\alpha))$ be a solution STC-RELAX(α). If $\alpha > \alpha^*$, then $\sum y_i(\alpha) = 0$. On the other hand, if $\alpha < \alpha^*$, then $\sum y_i(\alpha) > 0$ and $r(x(\alpha), y(\alpha), z(\alpha)) > \alpha$.*

Proposition 5.4 allows us to solve STC-RELAX with STC-RELAX(α) and a binary search, similar to STC-ILP. However, we can solve STC-RELAX directly with a single linear program, that is,

$$\begin{array}{lll}
\textsc{maximize} & \sum x_{ij} + \lambda \sum z_{ij} & \\
\textsc{subject to} & x_{ij} + z_{ij} \leq y_i & ij \in E \\
& x_{ij} + z_{ij} \leq y_j & ij \in E \\
& \sum y_i = 1 & \\
& x_{ij} + x_{jk} \leq y_j & (i, j, k) \in Z \\
& x_{ij}, z_{ij} \geq 0 & ij \in E \\
& y_i \geq 0 & i \in V
\end{array}$$

Proposition 5.5. *The LP given above solves STC-RELAX.*

Our LP is related to the LP proposed by Charikar [5], which is used to solve the densest subgraph problem exactly. We extend Charikar's LP by adding strong edges and additional STC constraints. Another related work is the LP proposed by Adriaens et al. [1] which provides a 2-approximation for MinSTC using similar wedge constraints.

Rounding phase: Next, we describe the heuristic used to obtain the subgraph and the labeling from the variables. Let (x^*, y^*, z^*) be the solution to STC-RELAX. First we define a collection of sets $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ where $S_j = \{i : y_i^* \geq y_j^*\}$. Then we enumerate over the collection of subgraphs induced by $\mathcal{S}$.

For each S_j, we initially set all the edges as weak. Then we enumerate over each edge $e \in E(S_j)$ starting from the largest z_e^*. Each edge e is labeled as strong if the STC property is not violated. This means that we check if there is any edge adjacent to any of the endpoints of e which is already labeled as strong and still creates a wedge with e. We continue the same process for all the edges $e \in E(S_j)$ in the descending order of its z_e^* value. Finally, out of all the subgraphs we pick the subgraph and the labeling that maximizes our score.

Constructing a labeling for a single S_j amounts to enumerating over the wedges in $O(nm)$ time, leading to a total time of $O(n^2 m)$ for the rounding.

5.3 Label, find the densest subgraph, and relabel

Next, we explain two algorithms that combine the existing methods for finding the densest subgraph and finding the STC-compliant labeling in an entire graph.

The approach is as follows. First, we label the edges of the entire graph using Apr-MinSTC (see Section 3). Then we construct a weighted version of the graph assigning a weight of 1 for strong edges and a weight of λ for weak edges. Next, we search for the densest subgraph using Cut or Peel (see Section 3) in the new weighted graph. Finally, we relabel *only* the subgraph induced by the returned solution. Relabeling is used to improve the score since some of the edges might be marked as weak since they contributed to certain wedges in the original graph, nevertheless, some edges no longer contribute to all of those wedges. The pseudo-code for this

Algorithm 1: STC-Cut(G, λ) and STC-Peel(G, λ), both find a subgraph U and a labeling L with good $q(U, L; \lambda)$.

1 $L \leftarrow$ Apr-MinSTC(G);
2 $H \leftarrow$ the weighted graph by setting 1 to strong edges and λ to weak edges of G;
3 $U \leftarrow$ Cut(H) [9] or Peel(H) [5];
4 $L' \leftarrow$ Apr-MinSTC$(G(U))$;
5 **return** subgraph U and its labeling L';

method is given in Algorithm 1. We call the algorithm as STC-Cut or STC-Peel based on the subroutine used in Line 3 of Algorithm 1.

Next, we present the computational complexities of the STC-Peel and STC-Cut algorithms.

Proposition 5.6. *Assume a graph G with n nodes and m edges. Assume that the wedge graph $Z(G)$ contains n' nodes and m' edges. Then the running time of STC-Peel is in*

$$O\left(\sum_{v \in V} \deg(v)^2 + (m + n \log n) + (m' + n')\right) \subseteq O(nm) \quad .$$

Proof. The number of wedges in G, and hence the number of edges in $Z(G)$ is in $O\left(\sum_{v \in V} \deg(v)^2\right) \subseteq O(n \sum_{v \in V} \deg(v)) \subseteq O(nm)$. The number of vertices, n', in the wedge graph $Z(G)$ is in $O(m)$. Apr-MinSTC estimates the minimum vertex cover problem with a maximum matching for $Z(G)$ and the subgraph, which has a running time of $O(n' + m')$ when the adjacency list representation is used for the graph [7]. We can execute Peel in $O(m + n \log n)$ time. The claim follows. □

Proposition 5.7. *Assume a graph G with n nodes and m edges. Assume that the wedge graph $Z(G)$ contains n' nodes and m' edges. Then the running time of STC-Cut is in*

$$O\big(mn + n(n + m) \log n + (m' + n')\big) \subseteq O(mn \log n) \quad .$$

Proof. The only change compared to Proposition 5.6 is that we are using an exact algorithm instead of an approximation algorithm for finding the densest subgraph. The exact algorithm for an edge-weighted graph takes $O(M(n, n + m) \log n)$ time, and $M(n, n + m)$ is the time taken to solve the min-cut problem for a graph with n number of nodes and $(n+m)$ number of edges. It takes $O(n(n + m))$ to find the minimum cut [18]. The claim follows. □

5.4 Peeling with continuous relabeling

The STC-Peel algorithm, given in the previous section, first finds a labeling and then uses Peel that constructs a set of subgraphs among which the subgraph with the highest score is selected. During this search, the labeling remains fixed. Our final algorithm modifies this approach by relabeling the graph as we are constructing the subgraphs.

Our approach is as follows. We start from the whole graph G and label the edges as either strong or weak using Apr-MinSTC. Given a labeling L and a subgraph U, let the weighted degree for a vertex $\deg_\lambda(v, U, L, \lambda)$ be defined as the sum of strong edges and weak edges in U incident to v weighted by λ, i.e., $\deg_\lambda(v, U, L) = \deg_s(v, U, L) + \lambda \deg_w(v, U, L)$ We drop L, U or λ when it is clear from the context. At each iteration, we delete the node that has the minimum weighted degree $\deg_\lambda(v)$. After removing each vertex we relabel the remaining set of edges. Finally, we choose the subgraph U which corresponds to the maximum score $q(U, \lambda)$ out of all the iterations. The naive version for this method is given in Algorithm 2.

Algorithm 2: Grd-Naive(G, λ), finds a subgraph U and a labeling L with good $q(U, L; \lambda)$

1 $U \leftarrow V$;
2 **while** there are nodes **do**
3     $L \leftarrow$ Apr-MinSTC$(G(U))$;
4     $u \leftarrow \arg\min_{v \in U} \deg_\lambda(v, U, L, \lambda)$;
5     $U \leftarrow U \setminus \{u\}$;
6 **return** best tested U and its labeling L;

Next, we explain several tricks to speed up the naive implementation of Algorithm 2. We focus on updating the wedge graph, modifying the minimum vertex cover, and updating individual scores of each vertex without computing them from scratch.

Maintain wedge graph: Note that on Line 3 of Algorithm 2, we need to repeatedly construct a wedge graph to solve MinSTC. We can avoid this by maintaining the existing wedge graph as vertices are deleted.

When a node is deleted we need to consider only *deleting* respective edges in the wedge graph since new wedges cannot be introduced. Note that an edge in the original graph G corresponds to a node in the wedge graph $Z(G)$ and edges in $Z(G)$ represent wedges in G. Next, we state how to maintain $Z(G)$ when a vertex is deleted in Proposition 5.8.

Proposition 5.8. *Let $G = (V, E)$ be a graph. Let v be a vertex in G. Define $G' = G(V \setminus \{v\}, E \setminus N(v))$, where $N(v)$ is the set of adjacent edges of vertex v in G. Then, a new wedge graph $Z(G')$ is formed by deleting the vertices in $Z(G)$ corresponding to $N(v)$.*

We omit the straightforward proof.

Dynamic vertex cover using maximal matching: Next, we consider updating the vertex cover after a vertex deletion.

Recall that we use maximum matching to approximate the vertex cover in Apr-MinSTC. Given a maximal matching of the current graph, Ivković and Lloyd [10] presented a simple algorithm to maintain the cover when an *edge* is deleted or inserted. Here we modify their algorithm slightly to adapt to a node deletion from G. Let us consider the case where the vertex v is deleted from the original graph G. Note that $N(v)$ is a set of edges in G which corresponds to a subset of nodes in $Z(G)$. According to Proposition 5.8, the set of nodes corresponding to $N(v)$ should be deleted from the wedge graph $Z(G)$ to compensate for the deleted vertex. We assume that a maximal matching M of $Z(G)$ is given.

The algorithm is as follows. We iterate over the elements in $N(v)$ and pick a node $a \in N(v)$ in $Z(G)$. We then test whether there is an edge (a, b) in M for some b. There can be only one, and if there is, we delete it. Upon such deletion, M may no longer be maximal since b may have a single adjacent edge that can be added. We search for such an edge and add it if one is found.

Algorithm 3: Dynamic-Vertex-cover(M, v), maintains a vertex cover (a maximal matching M) when a node v is deleted

foreach $a \in N(v)$ **do**
  if there is b such that $(a, b) \in M$ **then**
    delete (a, b) from M;
    if $b \notin N(v)$ **and** there is $t \notin N(v)$ such that t is not an endpoint of any edge in M and $(b, t) \in E(Z(G))$ **then**
      add (b, t) to M;
return M;

The pseudocode is given in Algorithm 3. Algorithm 3 still produces a maximal matching; thus a 2-approximation for Min-Vertex-cover is guaranteed.

Speeding the vertex selection: We can speed up finding the next vertex by maintaining $\deg_\lambda(v, \lambda)$ in a priority queue. Once a vertex is deleted, we need to update the degree of its neighboring nodes. Also, we may need to update the weighted degree of the affected vertices if the vertex cover of $Z(G)$ changes. However, the number of changed edges in the vertex cover is constant. The final version of the algorithm is presented in Algorithm 4.

Algorithm 4: Greedy(G, λ), finds a subgraph U and a labeling L with good $q(U, L; \lambda)$

$L \leftarrow$ Apr-MinSTC(G);
$P \leftarrow$ priority queue where each node is ranked by $\deg_\lambda(v, L)$;
$U \leftarrow V$;
while there are nodes **do**
  $u \leftarrow \arg\min_{v \in U} \deg_\lambda(v, U, L)$;
  $U \leftarrow U \setminus \{u\}$;
  Update the wedge graph $Z((U, E(U)))$;
  Update labeling L using Algorithm 3;
  Update P;
return best tested U and its labeling L;

Next, we state the computational complexity of Greedy.

Proposition 5.9. *Assume a graph G with n nodes and m edges. Assume that the wedge graph $Z(G)$ contains n' nodes and m' edges. Then the running time of* Greedy *is in*

$$O\big(mn + (n' + m') + m \log n + nm\big) \subseteq O(nm) \quad .$$

Proof. Let G_i be the graph at ith iteration. Consider deleting vertex u from G_i. Upon deletion, we need to update the priorities of the affected nodes in the queue.

When u is deleted from G_i, we need to delete the set of nodes in $Z(G)$ which corresponds to the adjacent edges of u. For each deleted vertex in $Z(G)$, there can be at most one adjacent edge that belongs to the existing matching set. Therefore, to compensate for the edge that is removed from the maximal matching set, we need to add at most one edge to the matching. The two endpoints of the newly added edge correspond to two edges in G_{i+1}. Therefore, the total number of vertices that require updating priorities is at most 4. Moreover, deleting one edge from the existing matching set will affect the priorities of at most 2 vertices.

In summary, $O(\deg u)$ nodes need to be updated when we delete u. Consequently, the total update time of P is in $O(m \log n)$. Moreover, the total update time for $Z(G)$ is in $O(n' + m')$. Updating M requires finding a new edge which may cost $O(n)$ time, consequently, updating M requires $O(nm)$ total time. Finally, the update time for $(U, E(U))$ is in $O(m)$.

Initially, constructing $Z(G)$ requires $O\big(\sum_{v \in V} \deg(v)^2\big) \subseteq O(nm)$ time and VC-Mat requires $O(n' + m') \subseteq O(nm)$ time.

Combining these times proves the claim. □

6 Experimental evaluation

Next, we evaluate our algorithms experimentally. We first generate a synthetic dataset with a dense subgraph component and test how well our algorithms perform. Next, we study the performance of the algorithms on real-world networks. We implemented the algorithms in Python[1] and performed the experiments using a 2.4GHz Intel Core i5 processor and 16GB RAM. In our experimental evaluation, we used Gurobi solver in Python to solve the ILPs and LPs associated with STC-ILP and STC-LP respectively.

Synthetic dataset: We will now explain how the synthetic dataset was generated. Given a vertex set V of size 230, we split V into dense and sparse components D and S. Here, we randomly selected D to have 38 nodes and S to have 192 nodes. We sampled the edges using a stochastic block model, with the edge probabilities being $p_d = 1$, $p_s = 0.3$, and $p_c = 0.05$ for dense component, sparse component, and cross edges, respectively. The resulting graph had 5 197 edges, and the wedge graph had 5 197 nodes and 179 100 edges. The density of D was $|E(D)|/|D| = 18.5$.

Results using synthetic dataset: We report our results in Table 1. First, we see that all our algorithms find the ground truth by achieving a score of 18.5 which is the density of our planted clique of size 38 for example when $\lambda = 0.4$ and $\lambda = 0.2$. Note that STC-ILP produced the results within an hour only for the $\lambda = 0.2$ case. Since STC-ILP solves an ILP in each round, it was inefficient to run for the other λ values and we stopped the execution after one hour.

As λ increases, our algorithms tend to find a score greater than 18.5 by deviating away from the planted clique. We also see that STC-LP which solves a linear program runs significantly slower than Greedy, STC-Peel, and STC-Cut algorithms.

Next, we study how the scores and the percentage of weak edges vary as a function of λ as shown in Figure 2. We can observe that both STC-Cut and STC-Peel produce equal scores whereas STC-Peel and STC-LP slightly underperform at $\lambda = 0.6$ and $\lambda = 0.5$ respectively as shown in Fig. 2a. Moreover, the STC-LP slightly outperforms other algorithms when $\lambda \geq 0.7$. In terms of percentages of weak edges, all three algorithms produced the same decreasing trends according to Fig. 2b. There are no weak edges in the subgraphs produced by any of the algorithms when $\lambda \leq 0.4$ since scores

[1] The source code is available at https://version.helsinki.fi/dacs/.

Table 1: Results of the experiments for synthetic and real datasets. Here, λ is the weight parameter, columns CT, PL, GR, LP, and IP represent STC-Cut, STC-Peel, Greedy, STC-LP, and STC-ILP, respectively, columns in q are the discovered scores, columns in $|E_s|\%$ give the percentages of *strong* edges in discovered subgraph, and columns in time give the computational time.

		q					$\|E_s\|\%$					time				
Dataset	λ	CT	PL	GR	LP	IP	CT	PL	GR	LP	IP	CT	PL	GR	LP	IP
Synthetic	0.8	21.28	21.28	21.27	21.68		9.45	15.96	9.43	17.2		11.72s	6.69s	6.94s	29.58s	
	0.6	16.57	16.57	18.5	17.37		9.45	15.86	100	17.07		11.29s	6.4s	7.11s	36.9s	
	0.4	18.5	18.5	18.5	18.5		100	100	100	100		9s	4.73s	7.85s	16.97s	
	0.2	18.5	18.5	18.5	18.5	18.5	100	100	100	100	100	7.02s	7.02s	7.98s	12.92s	9m5s
Cora	0.8	2.58	2.33	2.3	2.65	2.69	32.51	10.03	5.16	22.16	28.57	18.14s	1.1s	1m1s	2.41s	43.26s
	0.6	1.99	1.84	2.01	2.14	2.29	74.15	7.35	81.82	21.57	37.5	17.51s	0.92s	1m1s	2.8s	11m1s
	0.4	2	1.78	2	1.62	2.09	100	100	100	39.13	69.57	14.68s	0.94s	46.73s	2.98s	4m23s
	0.2	2	1.78	2	1.32	2.02	100	100	100	66.67	95.24	15.48s	0.88s	42.34s	2.79s	4m17s
Citeseer	0.8	4	3.99	3.87	4.12	4.2	6.54	17.95	8.89	19.44	27.78	13.17s	0.51s	1m25s	1.27s	18.32s
	0.6	3.11	3.08	2.99	3.35	3.5	6.93	18.64	5.45	20.37	28.16	15.44s	0.52s	1m18s	1.28s	17.54s
	0.4	2.22	2.16	2.5	2.65	2.82	16.07	24.49	100	23.3	32.94	14.55s	0.56s	56.86s	1.3s	38.62s
	0.2	1.5	1.5	2.5	1.89	2.5	100	100	100	35.29	100	15.05s	0.68s	50.06s	1.29s	1m6s
PGP	0.8	15.96	15.59	15.4	16.48	16.86	18.41	15.24	15.68	32.05	42.19	1m33s	5.95s	31m49s	22.39s	6m55s
	0.6	12.51	12.07	12	13.97	14.66	17.75	24.68	100	33.22	42.19	1m38s	6.67s	31m52s	20.52s	8m40s
	0.4	10.5	10.5	11	11.44	12.56	100	100	100	34.94	48.19	1m38s	6.42s	31m4s	20.33s	9m24s
	0.2	10.5	10.5	11	10.46	12	100	100	100	67.29	100	1m40s	6.28s	24m20s	27.82s	9m3s
Email-EU	0.8	22.17	22.17	22.12	22.44		2.14	2.17	1.22	7.09		52.5s	22.21s	2m29s	3m37s	
	0.6	16.78	16.76	16.68	17.3		2.14	2.09	1.26	6.88		54.26s	21.13s	2m33s	4m26s	
	0.4	11.4	11.38	11.24	12.2		2.35	2.28	1.44	7.12		55s	21.76s	2m25s	5m16s	
	0.2	6.02	5.97	8	7.16		3.06	2.88	100	7.52		57.63s	23.54s	2m21s	4m57s	
Facebook	0.8	62.26	63.33	62.02	61.48		5.71	6.19	4.16	12.68		1m22s	37.31s	5m29s	7m27s	
	0.6	47.79	47.94	47.31	49.84		5.71	6.19	4.16	12.49		1m23s	36.05s	5m39s	6m34s	
	0.4	33.54	33.38	33.5	36.3		6.26	5.92	100	12.75		1m21s	35.21s	5m32s	9m30s	
	0.2	19.27	19.36	33	23.18		6.76	6.92	100	13.46		1m22s	34.21s	5m22s	9m27s	
LastFM	0.8	11.97	12	11.94	12.28		4.72	5.58	3.77	15.19		1m33s	11.99s	24m54s	1m1s	
	0.6	9.16	9.19	8.98	9.67		4.72	5.34	5.29	13.53		1m33s	12.09s	24m45s	58.42s	
	0.4	6.47	6.43	6.5	7.09		6.21	5.76	100	14.3		1m32s	11.38s	23m10s	1m3s	
	0.2	3.46	3.52	6.5	4.71		4.38	4.83	100	15.98		1m32s	11.8s	17m28s	1m1s	

Table 2: Characteristics of real-world datasets. Here, $|V|$ and $|E|$ give the number of vertices and edges, $|V(Z)|$ and $|E(Z)|$ are the number of vertices and edges in the wedge graph, $d = |E(U)|/|U|$ gives the density of the densest subgraph, and d_c is the density induced by Max-Clique.

Dataset	$\|V\|$	$\|E\|$	$\|V(Z)\|$	$\|E(Z))\|$	d	d_c
Cora	2 708	5 278	5 151	47 411	3.14	2
Citeseer	3 264	4 536	4 192	23 380	4.91	2.5
PGP	10 680	24 316	23 568	270 433	19.07	12
Email-EU	986	16 064	16 063	866 833	27.57	8.5
Facebook	747	30 025	30 022	1 177 951	76.73	33.5
LastFM	7 624	27 806	27 775	557 781	14.79	7

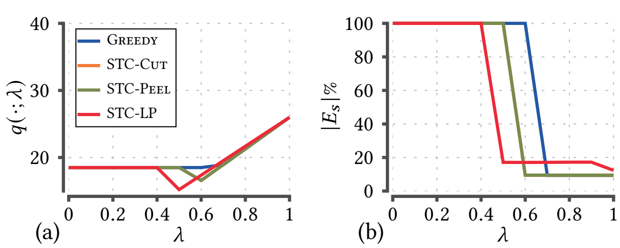

Figure 2: Scores and percentages of strong edges as a function of λ for *Synthetic* dataset.

are only contributed by the planted clique. Recall the connection to the maximum clique problem for $\lambda = 0$ from Proposition 4.1.

Finally, we study the running time as a function of the number of edges $|E|$ and the number of wedges $|V(Z)|$ in Figure 3. We randomly generated 6 datasets each with 5 000 nodes. The number of edges of the datasets uniformly ranges from 1×10^4 to 1.1×10^5. We see that STC-Cut and STC-Peel are the fastest while STC-LP is the slowest.

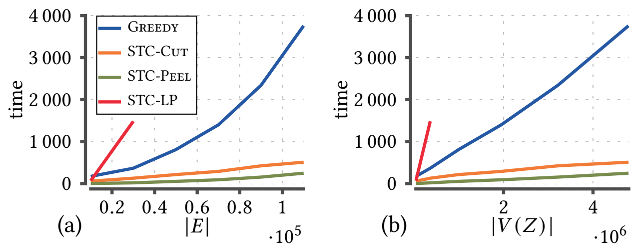

Figure 3: Time in seconds as a function of the number of edges $|E|$ and the number of wedges $|V(Z)|$.

Real-world datasets: We test our algorithms in publicly available, real-world datasets: *Cora* [20][2] and *Cite-seer* [20][2] datasets are citation networks. *Email-EU* is a collaboration network between researchers.[3] *Facebook* is extracted from a friendship circle of Facebook.[4] *LastFM* is a social network of LastFM users.[4] *PGP* is an interaction network of the users of the Pretty Good Privacy (PGP) algorithm.[5] The details of the datasets are shown in Table 2.

Results using real-world datasets: We present the results obtained using our algorithms in Table 1. We compare our algorithms in terms of scores, running time, and the percentage of the strong edges within subgraphs they returned for a set of λ values. Since STC-ILP invokes a sequence of integer programs, the algorithm does not scale for large datasets. We stopped the experiments that took over one hour. We always set $\epsilon = 0.01$ when testing each dataset with STC-ILP.

First, let us compare the scores across the algorithms. Our first observation is that STC-ILP always yields the highest score with the tested datasets while all other algorithms perform similarly in terms of scores: in most cases, they produce approximately equal scores. When STC-ILP is not usable, STC-LP has produced the maximum score except for 3 outlier cases where Greedy and STC-Peel algorithms obtained the maximum score 2 times and 1 time respectively. Nevertheless, for high λs all of them produce less deviated scores when compared to lower λs. As expected, q increases as λ increases for all 4 algorithms.

Next, let us look at column $|E_s|$% which gives the percentages of strong edges in the returned subgraph. Generally speaking, $|E_s|$% monotonically decreases as λ increases except for a few outlier cases. This is because when we assign a higher weight λ, it becomes more beneficial to include more weak edges.

Computation times are given in the *time* columns of Table 1. Greedy, STC-LP, and STC-Cut run significantly slower than STC-Peel. If we compare Greedy and STC-Cut, for all the tested cases STC-Cut runs faster despite having to solve a sequence of minimum cuts. This is due to the implementation differences as STC-Cut uses a fast native library to compute the minimum cuts.

Despite STC-ILP not being scalable for larger datasets, it runs faster than Greedy except for four cases with the tested datasets. For *Cora*, *Citeseer*, *PGP*, and *LastFM* datasets, STC-LP runs faster than all other algorithms except STC-Peel. However, for the two other remaining datasets, STC-LP is the slowest in comparison to the other three algorithms. We see that the running times are still reasonable in practice for the tested datasets; for example, we were able to compute the subgraph for the *Facebook* dataset, with over 30 000 edges and 1 000 000 wedges, in under ten minutes.

[2] https://networkrepository.com
[3] https://toreopsahl.com/datasets/
[4] http://snap.stanford.edu
[5] http://konect.cc/networks/arenas-pgp/

Table 3: Co-authorship case-study for *DBLP* dataset with weighted variant of STC-ILP. We set $\lambda = 0.8$ and $\epsilon = 0.01$. For each subgraph, we state the scores within brackets.

S1	P. S.Yu, C. C.Aggarwal, J.Han, W.Fan, J.Gao, X.Kong (6.00)
S2	C. H. Q.Ding, F.Nie, H.Huang, D.Luo (4.78)
S3	S.Yan, J.Yan, N.Liu, Z.Chen, H.Xiong, Q.Yang, Y.Fu, Y.Ge, H.Zhu, E.Chen, C.Liu, Q.Liu, B.Zhang (4.70)
S4	S.Lin, H.Hsieh, C.Li (4.23)
S5	C.Faloutsos, J.Sun, S.Papadimitriou, H.Tong, L.Akoglu, T.Eliassi-Rad, B.Gallagher (4.09)
S6	Y.Liu, M.Zhang, S.Ma, L.Ru (3.84)
S7	H.Liu, J.Tang, X.Hu, H.Gao (3.80)
S8	D.Phung, S.Venkatesh, S. KumarGupta, S.Rana, S.Tsumoto, S.Hirano (3.76)
S9	C.Böhm, I. S.Dhillon, C.Plant, C.Hsieh, P.Ravikumar (3.54)
S10	S.Günnemann, H.Kremer, T.Seidl, I.Assent, E.Müller, R.Krieger (3.46)

Case study: Next, we conducted a case study for *DBLP* [25][6] which contains co-authorship connections from top venues in data mining and machine learning (SDM, NIPS, ICDM, KDD, ECMLP-KDD, and WWW). Each node represents an author and each edge corresponds to a collaboration between two authors. We removed the author pairs who have less than 3 collaborations. The size of the dataset after prepossessing is $n = 4\,592$, $m = 5\,566$, and $|Z(G)| = 26\,073$. To compute a marginal weight that corresponds to an author pair, we assign a weight for each paper as one divided by the number of authors. We then weigh each edge (author-pair) by summing up the weights of all respective collaborations. Then we ran a weighted version of STC-ILP whose objective is to maximize the edge-weighted score,

$$\frac{\sum_{\text{strong } e \in E(U)} w(e) + \lambda \sum_{\text{weak } e \in E(U)} w(e)}{|U|} .$$

We found top-10 non-overlapping subgraphs iteratively by deleting the returned subgraph in each iteration and then considering the remaining graph to find the next subgraph. We set $\lambda = 0.8$ and $\epsilon = 0.01$. The list of author subgraphs is shown in Table 3. We see that the variant of STC-ILP discovered subgraphs of prolific authors.

7 Concluding remarks

We introduced a novel dense subgraph discovery problem that takes into account the strength of ties within the subgraph. Here we label each edge either as strong or weak based on the strong triadic closure principle (STC). The STC property requires that if one node strongly connects with two other nodes, then those

[6] https://www.aminer.org/citation

two nodes should at least have a weak connection between them. Our goal was to maximize a density-like measure defined as the sum of the number of strong edges and the number of weak edges weighted by a weight parameter, divided by the number of nodes within the subgraph. We showed that our optimization problem is **NP**-hard, and connects the two well-known problems of finding dense subgraphs and maximum cliques. To solve the problem, we presented an exact algorithm based on integer linear programming. In addition, we presented a polynomial-time algorithm based on linear programming, a greedy heuristic algorithm, and two other straightforward algorithms based on the algorithms for the densest subgraph discovery.

The experiments with synthetic data showed that our approach recovers the latent dense components. The experiments on real-world networks confirmed that our proposed algorithms discovered the subgraphs reasonably fast in practice. Finally, we presented a case study where our algorithm produced interpretable results suggesting the practical usefulness of our problem setting and algorithms.

The idea of combining the dense subgraph problem with the STC property opens up several lines of work. For example, instead of using the ratio between the number of edges and the number of nodes as the density measure, we can incorporate other density measures.

Acknowledgments

This research is supported by the Academy of Finland project MALSOME (343045).

References

[1] Florian Adriaens, Tijl De Bie, Aristides Gionis, Jefrey Lijffijt, Antonis Matakos, and Polina Rozenshtein. 2020. Relaxing the strong triadic closure problem for edge strength inference. *Data Mining and Knowledge Discovery* 34 (2020), 611–651.

[2] Chamalee Wickrama Arachchi and Nikolaj Tatti. 2023. Jaccard-constrained dense subgraph discovery. *arXiv preprint arXiv:2308.15936* (2023).

[3] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. 2000. Greedily finding a dense subgraph. *Journal of Algorithms* 34, 2 (2000), 203–221.

[4] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Charalampos Tsourakakis. 2015. Space-and time-efficient algorithm for maintaining dense subgraphs on one-pass dynamic streams. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 173–182.

[5] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *APPROX*. 84–95.

[6] Kenneth L Clarkson. 1983. A modification of the greedy algorithm for vertex cover. *Inform. Process. Lett.* 16, 1 (1983), 23–25.

[7] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.

[8] Werner Dinkelbach. 1967. On nonlinear fractional programming. *Management science* 13, 7 (1967), 492–498.

[9] Andrew V Goldberg. 1984. Finding a maximum density subgraph. (1984).

[10] Zoran Ivković and Errol L Lloyd. 1993. Fully dynamic maintenance of vertex cover. In *International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer, 99–111.

[11] Vinay Jethava and Niko Beerenwinkel. 2015. Finding dense subgraphs in relational graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 641–654.

[12] Narendra Karmarkar. 1984. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. 302–311.

[13] Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *International colloquium on automata, languages, and programming*. Springer, 597–608.

[14] Athanasios L Konstantinidis, Stavros D Nikolopoulos, and Charis Papadopoulos. 2018. Strong triadic closure in cographs and graphs of low maximum degree. *Theoretical Computer Science* 740 (2018), 76–84.

[15] Athanasios L Konstantinidis and Charis Papadopoulos. 2020. Maximizing the strong triadic closure in split graphs and proper interval graphs. *Discrete Applied Mathematics* 285 (2020), 79–95.

[16] Antonis Matakos and Aristides Gionis. 2022. Strengthening ties towards a highly-connected world. *Data mining and knowledge discovery* 36, 1 (2022), 448–476.

[17] Lutz Oettershagen, Athanasios L Konstantinidis, and Giuseppe F Italiano. 2022. Inferring Tie Strength in Temporal Networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 69–85.

[18] James B Orlin. 2013. Max flows in O (nm) time, or better. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 765–774.

[19] Victor Reis and Thomas Rothvoss. 2023. The subspace flatness conjecture and faster integer programming. *arXiv preprint arXiv:2303.14605* (2023).

[20] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *AAAI*. https://networkrepository.com

[21] Polina Rozenshtein, Nikolaj Tatti, and Aristides Gionis. 2017. Inferring the strength of social ties: a community-driven approach. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1017–1025.

[22] Alexander Schrijver. 1998. *Theory of Linear Integer Programming*. John Wiley & Sons.

[23] Konstantinos Semertzidis, Evaggelia Pitoura, Evimaria Terzi, and Panayiotis Tsaparas. 2019. Finding lasting dense subgraphs. *Data mining and knowledge discovery* 33, 5 (2019), 1417–1445.

[24] Stavros Sintos and Panayiotis Tsaparas. 2014. Using strong triadic closure to characterize ties in social networks. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1466–1475.

[25] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. 2008. Arnetminer: extraction and mining of academic social networks. In *KDD*. 990–998.

[26] Charalampos Tsourakakis. 2015. The k-clique densest subgraph problem. In *WWW*. 1122–1132.

[27] Jan van den Brand. 2020. A deterministic linear program solver in current matrix multiplication time. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 259–278.

[28] David Zuckerman. 2006. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*. 681–690.

A Appendix

A.1 Computational complexity proofs

Proof of Proposition 4.1. We will show the **NP**-hardness of stc-den by a reduction from the **NP**-hard Max-Clique problem. As $\lambda = 0$, we simply write the score as $q(U, L)$ instead of $q(U, L; \lambda) = q(U, L; 0)$ for brevity. Assume that the set U with the labeling L is an optimal solution to the stc-den problem with $\lambda = 0$, maximizing the score $q(U, L)$ while satisfying the STC property. The density of the strong edges in U is then

$$q(U, L) = \frac{\sum_{v \in U} \deg_s(v, U, L)}{2|U|}. \quad (7)$$

Consider a vertex w in U with the highest number $\deg_s(w, U, L)$ of strong edges connected to it. As the maximum number of strong edges, $\deg_s(w, U, L)$ has to be at least the average,

$$\deg_s(w, U, L) \geq \frac{\sum_{v \in U} \deg_s(u, U, L)}{|U|}. \quad (8)$$

Then for any two vertices v and u that have strong edges (v, w) and (u, w) connecting them to w, there must be an edge between v and u to satisfy the STC property.

Thus, the vertex w and its strong neighbors form a clique C with $\deg_s(w, U, L) + 1$ vertices. Consider only having these vertices in C and having a labeling L' that labels each edge in the clique as strong. This would satisfy the STC property and would give a score of

$$q(C, L') = \frac{|C|(|C| - 1)}{2|C|} = \frac{|C| - 1}{2} = \frac{\deg_s(w, U, L)}{2}.$$

From Equations 7 and 8, we get

$$q(C, L') = \frac{\deg_s(w, U, L)}{2} \geq \frac{\sum_{v \in U} \deg_s(v, U, L)}{2|U|} = q(U, L). \quad (9)$$

Thus, the clique C has at least the same score as U, which has the maximum score of all subgraphs, so $q(C, L') = q(U, L)$. This means that C must be a maximum size clique in the input graph G, as larger cliques would give a higher score than U.

Therefore, by finding an optimal set of vertices U and labeling L we can find a maximum clique C. Thus, STC-DEN is **NP**-hard. □

Proof of Proposition 4.2. Assume that we can find a set U with labeling L that is an α-approximation to STC-DEN, while the optimal solution has value $q(C^*, L^*)$ with a maximum clique C^* and labeling L^*. Consider then the vertex w with the highest number of strong edges and construct the clique C consisting of w and its strong neighbors. Define the labeling L' such that each edge in the clique C is labeled as strong. Using Equation 9 and that $q(U, L)$ is an α-approximation, we get

$$q(C, L') \geq q(U, L) \geq \alpha q(C^*, L^*).$$

But as the score of a clique C with only strong edges is $q(C, L') = \frac{|C|-1}{2}$, we have

$$\frac{|C| - 1}{2} \geq \alpha \frac{|C^*| - 1}{2}.$$

By solving for $|C|$ and using $\alpha \leq 1$, we get

$$|C| \geq \alpha \left(\left|C^*\right| - 1\right) + 1 \geq \alpha \left|C^*\right|.$$

This means that we have an α-approximation for MAX-CLIQUE. Therefore, any inapproximability results for MAX-CLIQUE also apply for the $\lambda = 0$ case of STC-DEN. Using the result by Zuckerman [28] then finishes the proof. □

To prove Proposition 4.3 we need the following lemma.

Lemma A.1. *Assume graph G. Let $X \subsetneq Y$ be two subgraphs, and let L be a labeling defined on Y. Define*

$$\Delta(X, Y) = \frac{m_s(Y) + \lambda m_w(Y) - m_s(X) - \lambda m_w(X)}{|Y| - |X|} \quad .$$

If $\Delta(X, Y) < q(Y, L)$, then $q(Y, L) < q(X, L)$, if $\Delta(X, Y) > q(Y, L)$, then $q(Y, L) > q(X, L)$, and if $\Delta(X, Y) = q(Y, L)$, then $q(Y, L) = q(X, L)$.

Proof. Assume $\Delta(X, Y) < q(Y, L)$. Multiply by $(|Y| - |X|)|Y|$ and subtract $|Y|(m_s(Y) + \lambda m_w(Y))$ from both sides. Dividing by $-|X||Y|$ then gives $q(X, L) > q(Y, L)$, proving the first claim. The proofs for other claims are identical. □

Proof of Proposition 4.3. We will prove the hardness by reducing an **NP**-hard problem MINSTC to our problem. In MINSTC, we are asked to label the full graph and minimize the number of weak edges [24]. Assume a graph G with nodes $V = v_1, \ldots v_n$. We assume that $n \geq 5$. We define a new graph H that consists of G and $k = \lceil 1/\lambda \rceil (n + 1)/2$ cliques C_i of size n. Let $\left\{c_{ij}\right\}$ be the nodes in C_i. For each j and i, we connect v_j with c_{ij}.

Let U be the optimal subgraph of H for STC-DEN and L be its labeling. Let L' be the labeling where every $E(C_i)$ is strong and the remaining edges are weak. Note that $q(U) \geq q(C_i, L') = (n-1)/2 \geq 2$ for any C_i. We claim that U contains every node in H.

To prove the claim, let us define $W_i = U \cap C_i$. If $|W_i| = 1, 2$, then $\Delta(U \setminus W_i, U) \leq 3/2 < q(U)$ and Lemma A.1 states that we can delete W_i from U and obtain a better score. Assume $3 \leq |W_i| < n$. Let $c \in C_i \setminus W_i$. We can safely assume that the edges between W_i and G are weak; otherwise, we can relabel them as weak and compensate by labeling any weak edge in W_i as strong. Now we can extend the labeling L to c by setting the edges from c to W_i as strong, and the possibly remaining edge as weak. We can show that $\Delta(U \setminus W_i, U) < \Delta(U, U \cup \{c\})$. Lemma A.1, applied twice, states that either deleting W_i or adding c improves the solution. Therefore, either $W_i = \emptyset$ or $W_i = C_i$.

Assume $W_i = C_i$ and $W_j = \emptyset$. The optimal labeling must be such that all edges between W_i and G are weak and the edges in W_i are all strong. We can extend the same labeling scheme to C_j. Then $\Delta(U \setminus C_i, U) = \Delta(U, U \cup C_j)$. If $\Delta(U, U \cup C_j) > q(U)$, Lemma A.1 implies that we improve the solution by adding C_j, which is a contradiction. Hence, $\Delta(U \setminus C_i, U) \leq q(U)$. Lemma A.1 implies that we can safely delete C_i. Applying this iteratively we arrive to an optimal solution with nodes only in V. This cannot happen since then $q(U) \leq (n - 1)/2$, but then $q(C_i \cup V, L') = (n - 1)/2 + \lambda/2 > q(U)$. Therefore, $W_i = C_i$ for every i.

Finally, assume $v_j \notin U$. Then $\Delta(U, U \cup \left\{v_j\right\}) \geq \lambda k \geq (n+1)/2 > (n - 1)/2 + \lambda \geq \Delta(U \setminus C_i, U)$. Lemma A.1 states that either deleting any C_i or adding v_j improves the solution. This contradicts the optimality of U, so every $v_j \in U$.

Consequently, $V \subseteq U$. The optimal labeling must have every edge in C_i as strong, the cross-edges between C_i and G as weak, and the labels for edges in G solve MINSTC. □

A.2 Proofs for Section 5.1

Proof of Proposition 5.1. Let us write $f(U, L) = m_s(U, L) + \lambda m_w(U, L)$. Note that

$$f(U(\alpha), L(\alpha)) - \alpha|U(\alpha)| \geq 0. \quad (10)$$

Assume $\alpha > \alpha^*$. If $U(\alpha) \neq \emptyset$, then Eq. 10 implies that

$$q(U(\alpha), L(\alpha)) \geq \alpha > \alpha^*,$$

which contradicts the optimality of α^*. Thus, $U(\alpha) = \emptyset$.

Assume $\alpha < \alpha^*$. Then

$$\begin{aligned} f(U(\alpha), L(\alpha)) - \alpha|U(\alpha)| &\geq f(U^*, L^*) - \alpha\left|U^*\right| \\ &> f(U^*, L^*) - \alpha^*\left|U^*\right| = 0. \end{aligned}$$

That is, $f(U(\alpha), L(\alpha)) > \alpha|U(\alpha)|$, implying in turn that $U(\alpha) \neq \emptyset$ and $q(U(\alpha), L(\alpha)) > \alpha$. □

Proof of Proposition 5.2. Let L and U be the values of the interval when binary search is terminated. Note that $\alpha \geq L$ due to Proposition 5.1. We know that $U - L \leq \epsilon L$ and $L \leq \alpha^* \leq U$. Thus, $\alpha^* - L \leq U - L \leq \epsilon L$, or $\alpha^* \leq (1 + \epsilon)L \leq (1 + \epsilon)\alpha$. □

Proof of Proposition 5.3. Let α be the score of the solution X, L returned by STC-ILP, and let α^* be the score of the optimal solution X^*, L^* for STC-DEN. We will show that if $\alpha < \alpha^*$, then $\alpha^* - \alpha \geq 1/(bn^2)$, which contradicts with the fact that $\alpha^* - \alpha \leq \epsilon\alpha < \epsilon n/2 = 1/(bn^2)$.

To prove the claim, let $\Delta = \alpha^* - \alpha$. Then

$$\begin{aligned}\Delta &= \frac{m_s(X^*) + \frac{a}{b} m_w(X^*)}{|X^*|} - \frac{m_s(X) + \frac{a}{b} m_w(X)}{|X|} \\ &= \frac{|X|(bm_s(X^*) + am_w(X^*)) - |X^*|(bm_s(X) + am_w(X))}{b|X||X^*|}.\end{aligned}$$

Note that the numerator and the denominator are both integers. Consequently, if $\Delta > 0$, then $\Delta \geq 1/(bn^2)$. It follows that if we set $\epsilon = \frac{2}{bn^3}$, then STC-ILP finds the optimal solution in $O(\log n + \log b)$ number of rounds. □

A.3 Proofs for Section 5.2

Proof of Proposition 5.4. Scaling (x^*, y^*, z^*) by any constant $c > 0$ does not change the value of $r(\cdot, \cdot, \cdot)$ nor does it change the validity of the constraints in Eqs. 2–4. Therefore, we can safely assume that $x_e^*, z_e^* \leq 1$ and $y_i^* \leq 1$ and $i \in V$, for any $e \in E$ and $i \in V$. The claim now follows by repeating the steps of the proof of Proposition 5.1. □

Proof of Proposition 5.5. Scaling (x^*, y^*, z^*) by any constant $c > 0$ does not change the value of $r(\cdot, \cdot, \cdot)$ nor does it change the validity of the constraints in Eqs. 2–4. Therefore, we can safely require that $\sum y_i = 1$, which immediately proves the claim. □