ELTE Eötvös Loránd University

Faculty of Science

Searching and generating sparse (sub)graphs

Math project III. report

Bence Deák

Applied mathematics MSc

Supervisor:

Péter Madarasi

Research fellow

HUN-REN Alfréd Rényi Institute of Mathematics, and
Department of Operations Research, ELTE

Budapest
2025

1 Introduction

Throughout this report, we fix non-negative integers k and ℓ with $\ell < 3k$. In the *classical range* $\ell < 2k$, the graph $G = (V, E)$ is called (k,ℓ)-*sparse* if, for each subset $X \subseteq V$, the number $i_G(X)$ of edges induced by X satisfies $i_G(X) \le \max\{k|X| - \ell, 0\}$. In the extended range $2k \le \ell < 3k$, we require this inequality only for subsets with size at least three. If G is (k,ℓ)-sparse and has exactly $\max\{k|V| - \ell, 0\}$ edges, then it is called (k,ℓ)-*tight*. A graph is (k,ℓ)-*spanning* if it contains a (k,ℓ)-tight subgraph spanning all vertices. A (k,ℓ)-*block* of a (k,ℓ)-sparse graph is a subset $X \subseteq V$ that induces a (k,ℓ)-tight subgraph, and a (k,ℓ)-*component* is an inclusion-wise maximal (k,ℓ)-block. These notions form the basis of a rich combinatorial theory closely tied to matroids and rigidity, and they naturally give rise to the optimization and recognition problems studied in this work.

In previous semesters, we gave the first algorithm for the maximum-weight (k,ℓ)-sparse subgraph problem with a provably $O(n^2+m)$ running time, resolving a long-standing open question in combinatorial optimization and rigidity theory. As a direct corollary, the maximum-weight rigid subgraph problem — corresponding to the well-known case $k = 2$, $\ell = 3$ — can now be solved in quadratic time. Beyond its standalone significance, the maximum-weight (k,ℓ)-sparse subgraph problem also frequently arises as a subroutine in more complex combinatorial optimization problems. For example, our algorithm substantially improves the running time of approximation algorithms for the minimum-weight redundantly rigid and globally rigid subgraph problems [1] and their generalizations [2] in the metric case. Further applications include enumerating non-crossing minimally rigid frameworks [3], as well as recognizing kinematic joints [4].

During this semester, we considered the recognition problem for (k,ℓ)-sparse graphs. We developed three specialized algorithms for the parameter ranges $\ell \le k$, $k < \ell < 2k$, and $2k \le \ell < 3k$, significantly improving upon the previously best-known running times.

2 Recognizing (k,ℓ)-sparse graphs

In this section, we consider the range $0 \le \ell < 3k$, and address the recognition problem: given a graph $G = (V, E)$, decide whether G is (k,ℓ)-sparse. We assume $m \le kn$; in the range $2k \le \ell < 3k$, we additionally assume that G is simple. We first introduce the running-time notation and subroutine assumptions that underpin our analyses, then collect the common preliminaries used across parameter ranges. Building on these, we develop the algorithms for the ranges $\ell \le k$, $k < \ell < 2k$, and $2k \le \ell < 3k$, achieving significant asymptotic improvements over prior methods.

2.1 Running-time primitives and complexity overview

The recognition algorithms developed in the following subsections rely on several standard graph-theoretic subroutines as black boxes. Since their running times heavily impact our overall complexity bounds, we introduce concise notation for them — summarized in Table 1. Throughout, fix a positive constant K large enough so that every instance we consider henceforth of "size" n has at most Kn vertices, Kn edges, and (where relevant) total capacity at most Kn. The stated bounds reflect the best currently known algorithms.

Table 1: Notation and running-time bounds for standard subroutines.

Notation	Meaning	Best bound
$T_{MF}(n)$	Running time for computing a maximum integer flow in a network with integer arc capacities, where the number of vertices, the number of arcs, and the sum of arc capacities are each at most Kn.	$O(n^{1+o(1)})^*$ [5]
$T_{FD}(n,\kappa)$	Running time for decomposing a graph into κ forests, where the number of vertices and the number of edges are both at most Kn.	$O(n^{1+o(1)})^*$ [6]
$T_{RC}(n,\eta)$	Running time for determining rooted η-arc-connectivity in a rooted digraph with at most Kn vertices and edges.	$\begin{cases} O(n) & \text{if } \eta \le 2 \text{ [7, 8]} \\ O(n \log n) & \text{if } \eta > 2 \text{ [9]} \end{cases}$

* If we restrict ourselves to purely combinatorial algorithms, then the best bounds for the first and second rows change to $O(n\sqrt{n})$ [10, 11] and $O(n\sqrt{n \log n})$ [12], respectively.

A convenient regularization. For technical convenience, we also define

$$T'_{RC}(n,\eta) = n \cdot \max\left\{ \frac{T_{RC}(q,\eta)}{q} : q = 1, \ldots, n \right\}.$$

By construction, $T_{RC}(n,\eta) \le T'_{RC}(n,\eta)$ for all n, hence $T_{RC}(n,\eta) = O(T'_{RC}(n,\eta))$. Moreover, T'_{RC} satisfies the same asymptotic bounds as T_{RC} listed in Table 1.

Note 1. In the constructions used in this paper, the choice $K = 4k$ is adequate as a rough estimate. Note, however, that the asymptotic bounds above do not depend on the specific choice of K. Thus, throughout our constructions, we only establish that any quantity we bound by Kn — such as the number of vertices, the number of edges, or the total capacity — is $O(n)$.

Note 2. The most efficient known algorithm for the maximum flow problem [5] is based on an interior-point method, and although it has nearly linear time complexity in theory, it is very difficult to use in practice. A purely combinatorial alternative is Dinic's algorithm [10], whose running time is $O(n\sqrt{n})$ in the special case when the total capacity is $O(n)$ [11].

Note 3. For computing a forest decomposition, the fastest algorithm currently known [6] uses the nearly linear maximum-flow subroutine mentioned above. As a purely combinatorial alternative, one may use the Gabow-Westermann algorithm [12], which runs in time $O(n\sqrt{n \log n})$.

Note 4. We can check rooted η-arc-connectivity in a digraph with $O(n)$ vertices and $O(n)$ arcs as follows:

- When $\eta = 0$, there is nothing to check.
- When $\eta = 1$, perform a single graph search in linear time to test whether every vertex is reachable from s.
- When $\eta = 2$, use Tarjan's algorithm [7], which runs in $O(n)$ once the dominator tree of the digraph is built in linear time [8].
- When $\eta \ge 3$, use Gabow's algorithm with running time $O(n \log n)$ [9].

Summary of running times. Table 2 summarizes the asymptotic running times of the best previously known recognition algorithms and our new methods, developed later in this section, for the three parameter ranges. The bounds are expressed using the running-time primitives introduced above. For comparison, the rightmost column also lists the asymptotic bounds obtained by instantiating these primitives with the fastest known implementations.

Table 2: Asymptotic running-time bounds for the recognition problem.

Range	Old bounds	New bounds
$0 \le \ell \le k$	$O(n\sqrt{n \log n})$ [12]	$O(T_{RC}(n,\ell) + T_{MF}(n)) \subseteq O(n^{1+o(1)})^*$
$k < \ell < 2k$	$O(n^2)^\dagger$ [13]	$O(T_{FD}(n,k) + T'_{RC}(n,\ell-k)\log n) \subseteq O(n^{1+o(1)})^*$
$2k \le \ell < 3k$	$\begin{cases} O(n^2) & \text{if } \ell = 2k \text{ [14]} \\ O(n^3) & \text{if } \ell > 2k \text{ [14]} \end{cases}$	$O\big(n \cdot T_{RC}(n, \ell+1-2k)\big) \subseteq \begin{cases} O(n^2) & \text{if } \ell \le 2k+1 \\ O(n^2 \log n) & \text{if } \ell > 2k+1 \end{cases}$

* Under purely combinatorial implementations of our primitives, the new bounds become $O(n\sqrt{n})$ for the $\ell \le k$ case and $O(n\sqrt{n \log n})$ for the $k < \ell < 2k$ case.

† In the special case of Laman graphs, i.e., for $(k,\ell) = (2,3)$ with $m = 2n-3$, a recognition algorithm with running time $O(T_{FD}(n,2) + n \log n) \subseteq O(n^{1+o(1)})$ was developed in [15]. For the planar Laman case, a more specialized algorithm was proposed in [16] with a time complexity of $O(n \log^3 n)$.

2.2 Structural and algorithmic preliminaries

In this subsection, we develop the structural properties and algorithmic ingredients common to all parameter ranges. A central theme is a key lemma stating that an essential subtask of the recognition problem reduces to testing rooted η-arc-connectivity; since for small fixed η, this can be performed very efficiently (see Table 1), the subtask itself also admits a fast solution. We begin with a basic definition.

Definition 1. Given a digraph $D = (V, A)$ and an independent vertex set $U_0 \subseteq V$, we say that D is U_0-source if $\varrho_D(v) = 0$ for all $v \in U_0$.

Lemma 1. *Let k, ℓ, and t be natural numbers with $k > 0$ and $tk \le \ell \le (t+1)k$. Let $G = (V, E)$ be a graph, and let $U_0 \subseteq V$ be an independent set with size t. Then the following are equivalent:*

(i) The (k,ℓ)-sparsity condition holds for every vertex set that strictly contains U_0.

(ii) *G admits a k-indegree-bounded U_0-source orientation D_0, and for every such D_0, the digraph D_0' obtained by adding a root s, deleting U_0, and, for each $v \in V \setminus U_0$, adding $k - \varrho_{D_0}(v)$ parallel arcs from s to v, is rooted $(\ell - tk)$-arc-connected.*

(iii) *G admits a k-indegree-bounded U_0-source orientation D_0 such that the corresponding D_0' is rooted $(\ell - tk)$-arc-connected.*

Proof. We prove the implication cycle $(i) \Rightarrow (ii) \Rightarrow (iii) \Rightarrow (i)$. The step $(ii) \Rightarrow (iii)$ is immediate.

$(iii) \Rightarrow (i)$: Assume D_0 is a suitable orientation and the corresponding D_0' is rooted $(\ell - tk)$-arc-connected. Let X be any vertex set that strictly contains U_0 and pick any $u \in X \setminus U_0$. By Menger's theorem, there exist $q = \ell - tk$ arc-disjoint paths from s to u in D_0', say $P_1, \ldots, P_q$. Consider the orientation D' obtained from D_0 by deleting the first arc of each P_i and reversing the resulting paths. By construction, D' remains k-indegree-bounded, with $\varrho_{D'}(u) \leq (t+1)k - \ell$ and $\varrho_{D'}(v) = 0$ for each $v \in U_0$. Hence,

$$i_G(X) \leq \sum_{v \in X} \varrho_{D'}(v) = \varrho_{D'}(u) + \sum_{v \in X \setminus (U_0 \cup \{u\})} \varrho_{D'}(v) \leq (t+1)k - \ell + (|X| - (t+1))k = k|X| - \ell,$$

so the sparsity bound holds for X.

$(i) \Rightarrow (ii)$: Assume the sparsity bound holds for every vertex set strictly containing U_0. To prove the existence of a suitable orientation D_0, we apply the Orientation Lemma [17]. It states that for a given upper-capacity function $g : V \to \mathbb{N}$, there exists a g-indegree-bounded orientation of G if and only if

$$i_G(X) \leq \sum_{v \in X} g(v) = g(X)$$

holds for every vertex set $X \subseteq V$. Define g by setting $g(v) = 0$ for $v \in U_0$ and $g(v) = k$ for $v \notin U_0$. Fix any X and let $r = |X \setminus U_0|$. If $r = 0$, then $i_G(X) = 0 = g(X)$ since U_0 is independent. Otherwise, by the sparsity condition,

$$i_G(X) \leq i_G(X \cup U_0) \leq (t+r)k - \ell = rk + (tk - \ell) \leq rk = k \cdot |X \setminus U_0| + 0 \cdot |X \cap U_0| = g(X).$$

Hence the desired orientation D_0 exists.

We show that, for any suitable orientation D_0, the digraph D_0' constructed from it is rooted $(\ell - tk)$-arc-connected. Let $X \subseteq V \setminus U_0$ be any non-empty vertex set, and let q be the number of edges in G between X and U_0. In D_0, each such edge is oriented from U_0 into X. Hence,

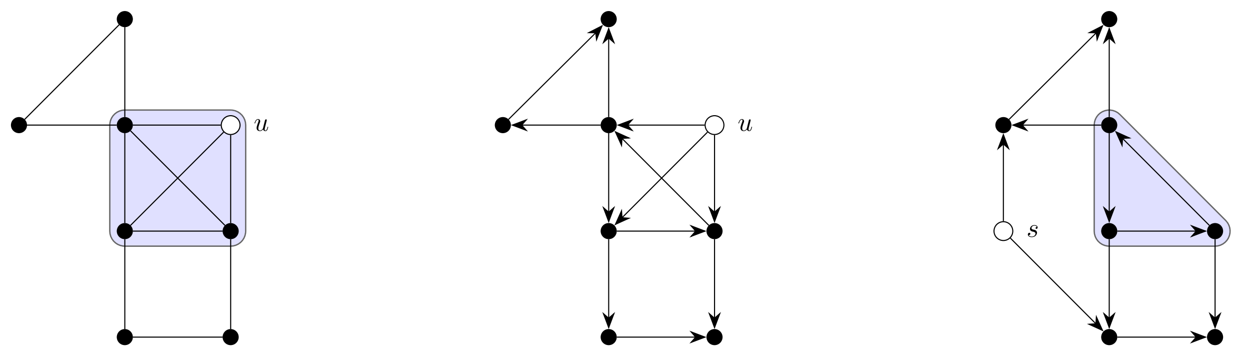
$$\sum_{v \in X} \varrho_{D_0'}(v) = \sum_{v \in X} \varrho_{D_0}(v) + \sum_{v \in X} (k - \varrho_{D_0}(v)) - q = k|X| - q.$$

Since U_0 is independent, we have $i_G(X \cup U_0) = i_G(X) + q$. Then by the sparsity bound,

$$\varrho_{D_0'}(X) = \sum_{v \in X} \varrho_{D_0'}(v) - i_G(X) = \sum_{v \in X} \varrho_{D_0'}(v) - (i_G(X \cup U_0) - q) \geq (k|X| - q) - (k(|X| + t) - \ell - q) = \ell - tk.$$

Therefore, D_0' is rooted $(\ell - tk)$-arc-connected. ∎

Example 1. Let $(k, \ell) = (2, 3)$ and consider the graph G on the left of Figure 1 with $U_0 = \{u\}$. The shaded four-vertex set X induces $6 > 5 = 2 \cdot 4 - 3$ edges, violating the (k, ℓ)-sparsity bound. The middle subfigure shows an orientation D_0 of G that satisfies the conditions of Lemma 1. From this, we construct the digraph D_0' with root s as prescribed. As seen in the right subfigure, the set $X \setminus U_0$ violates rooted arc connectivity in D_0'.



(a) The graph G and a vertex set that violates $(2, 3)$-sparsity.

(b) An orientation D_0 that satisfies the conditions of Lemma 1.

(c) The digraph D_0' and a vertex set that violates its rooted arc connectivity.

Figure 1: Application of Lemma 1.

Let U_0 be an independent set and let D_0 be a k-indegree-bounded U_0-source orientation of G. By Lemma 1, the question of whether there exists a violating vertex set strictly containing U_0 reduces to a rooted arc-connectivity problem. This yields the following algorithm:

Algorithm 1 Superset Sparsity

Input: A vertex set $U_0 \subseteq V$ with t elements, and a k-indegree-bounded U_0-source digraph $D_0 = (V, A)$.
Require: The inequalities $tk \leq \ell \leq (t+1)k$ are satisfied.
Output: `true` if there is no vertex set that strictly contains U_0 and violates the (k, ℓ)-sparsity of the underlying graph of D_0; `false` otherwise

procedure $\textsc{SupersetSparsity}_{k,\ell}(D_0 = (V, A), U_0)$
    $D_0' \leftarrow D_0 \setminus U_0 \cup \{s\}$ ▷ Initialize the digraph D_0' of Lemma 1 with root s
    for $v \in V \setminus U_0$ **do**
        for $i \leftarrow 1, \ldots, k - \varrho_{D_0}(v)$ **do**
            $D_0' \leftarrow D_0' \cup \{sv\}$ ▷ Insert an arc from s to v in D_0'
        end for
    end for
    if D_0' is rooted $(\ell - tk)$-arc-connected **then** ▷ Use the reduction described in Lemma 1
        return `true`
    else
        return `false`
    end if
end procedure

Proposition 1. *Algorithm 1 runs in $O(T_{RC}(n, \ell - tk))$ time.*

Proof. We construct D_0' in linear time. Since D_0' has $O(n)$ vertices and $O(n)$ arcs, it takes $T_{RC}(n, \ell - tk)$ time to decide whether it is rooted $(\ell - tk)$-arc-connected. Therefore, the total running time of the algorithm is

$$O(T_{RC}(n, \ell - tk) + n) = O(T_{RC}(n, \ell - tk)).$$

■

For Lemma 1 and Algorithm 1 to be applicable, we first require a k-indegree-bounded orientation. By the Orientation Lemma [17], such an orientation always exists in a (k, ℓ)-sparse graph; the question is how efficiently we can find one. The following lemma gives the answer.

Lemma 2. *There is an $O(T_{MF}(n))$-time algorithm that, for every input graph $G = (V, E)$, either produces a k-indegree-bounded orientation of G or certifies that no such orientation exists.*

Proof. We reduce the problem to a maximum-flow instance. First, take an arbitrary orientation $D = (V, A)$ of G. Our goal is to obtain a k-indegree-bounded orientation by reversing some arcs of D. The choice can be encoded by a vector $x : A \to \{0, 1\}$ with $x(e) = 1$ if we reverse e and $x(e) = 0$ otherwise. After applying the reversals prescribed by x, the indegree of a vertex $u \in V$ becomes $\varrho_D(u) - \varrho_x(u) + \delta_x(u)$. Hence, x is feasible exactly when, for each u,

$$\varrho_D(u) - \varrho_x(u) + \delta_x(u) \leq k \iff \varrho_x(u) - \delta_x(u) \geq \varrho_D(u) - k =: b(u).$$

So far, the upper bound for each $\varrho_x(u) - \delta_x(u)$ is infinite. We can ensure that the sum of the absolute values of the lower and upper bounds in our inequality system is finite — in fact, $O(n)$ — by adding the redundant constraint

$$\varrho_x(u) - \delta_x(u) \leq d_G(u)$$

for every vertex u. This does not change the set of feasible solutions since $\delta_x(u)$ is non-negative and $\varrho_x(u) \leq d_G(u)$.

The problem defined by the inequalities above reduces directly to a feasible-circulation instance. Give every arc of D lower bound 0 and upper bound 1. Add a new vertex r and, for each u, insert an arc from u to r with lower bound $b(u)$ and upper bound $d_G(u)$. Clearly, the original feasibility problem is equivalent to finding a feasible integer circulation in the constructed network.

We now reduce the obtained circulation problem to a maximum-flow instance in the standard way [18, p. 120]. For each arc e, let $f(e)$ denote its lower bound, and define $\varphi : V \to \mathbb{Z}$ by

$$\varphi(v) = \varrho_f(v) - \delta_f(v).$$

Decrease both the lower and upper bounds of every arc e by $f(e)$ so that all lower bounds become 0. Introduce a source s and a sink t, then scan over all other vertices: for each vertex v, if $\varphi(v) > 0$, add an arc from s to v with capacity $\varphi(v)$, otherwise add an arc from v to t with capacity $-\varphi(v)$. The original feasible-circulation instance is solvable if and only if, in the obtained network, there exists a feasible s-t flow that saturates every arc leaving s. After finding such a flow, the solution to the circulation problem is easy to reconstruct: delete s and t, then increase the flow value of each arc e by $f(e)$. Since the sum of the absolute values of the arc capacities in the feasible-circulation instance is $O(n)$, the same holds for the maximum-flow network. Hence, the asymptotic running time of the maximum-flow computation is $T_{MF}(n)$. Combined with the linear time needed to construct the flow network and to recover the orientation from the computed saturating flow, the overall running time is

$$O(T_{MF}(n) + n) = O(T_{MF}(n)).$$

■

To apply Lemma 1, we require a k-indegree-bounded orientation that is also U_0-source. We show that, if such an orientation exists, it can be obtained from any k-indegree-bounded orientation via augmenting paths. The next lemma presents the key idea behind this approach.

Lemma 3. *Let $G = (V, E)$ be a graph with a k-indegree-bounded orientation $D = (V, A)$, and let $U_0 \subseteq V$. If D is not U_0-source and there is no directed path in D from $S = \{v \in V \setminus U_0 : \varrho_D(v) < k\}$ to U_0, then G admits no k-indegree-bounded U_0-source orientation.*

Proof. Suppose that D is not U_0-source and there is no path in D from S to U_0, yet G admits a k-indegree-bounded U_0-source orientation D_0. Let T be the set of vertices from which at least one vertex of U_0 is reachable in D. Then $\varrho_D(T) = 0$ and $\varrho_D(v) = k$ for each $v \in T \setminus U_0$. Hence,

$$k|T \setminus U_0| \geq \sum_{v \in T} \varrho_{D_0}(v) \geq i_G(T) = \sum_{v \in T} \varrho_D(v) > \sum_{v \in T \setminus U_0} \varrho_D(v) = k|T \setminus U_0|,$$

which is a contradiction. ■

The lemma above also yields an augmenting-path algorithm for finding a suitable orientation D_0, starting from any k-indegree-bounded orientation D. In each step, we find a path from S to U_0 and reverse its arcs, thereby decreasing the sum of indegrees in U_0. If there is no path from S to U_0, then either the current orientation is already suitable or no such orientation exists. We give the exact implementation below.

Algorithm 2 Reorient

Input: A k-indegree-bounded digraph $D = (V, A)$, and a vertex set $U_0 \subseteq V$ with t elements.
Output: A k-indegree-bounded U_0-source reorientation D_0 of D if it exists; $\varnothing$ otherwise.

procedure REORIENT$_{k,\ell}(D = (V, A), U_0)$
  while D is not U_0-source **do**
    find a path P in D from $\{v \in V \setminus U_0 : \varrho_D(v) < k\}$ to U_0
    if no such path exists **then**
      return $\varnothing$ ▷ No suitable orientation exists
    end if
    reverse the arcs of P in D ▷ Decrease the sum of indegrees in U_0
  end while
  return D
end procedure

Proposition 2. *Algorithm 2 runs in $O(tn)$ time.*

Proof. Since the original digraph D is k-indegree-bounded and $|U_0| = t$, we have

$$\sum_{v \in U_0} \varrho_D(v) \leq |U_0| \cdot k = tk.$$

Reversing an augmenting path decreases the total indegree in U_0 by 1, so the algorithm makes at most tk iterations. As each augmenting path is found in $O(n)$ time via a single graph traversal, the overall running time is $O(tn)$. ■

Note 5. Throughout, we treat t as a constant whenever the procedure is invoked; this way, the time complexity of Algorithm 2 becomes $O(n)$.

2.3 The range $\ell \leq k$

The case $\ell \leq k$ is most commonly handled in the literature [13] by the Gabow-Westermann algorithm [12] with running time $O(n\sqrt{n \log n})$, using matroid partition techniques. By applying Lemmas 1 and 2, we can give a more efficient solution as follows.

Algorithm 3 Check Sparsity for $\ell \leq k$

Input: An undirected graph $G = (V, E)$.
Output: `true` if G is (k, ℓ)-sparse; `false` otherwise.

---

procedure $\textsc{CheckSparsity}_{k,\ell}(G = (V, E))$
    $D \leftarrow$ a k-indegree-bounded orientation of G ▷ Use Lemma 2
    if there is no such D **then**
        return `false`   ▷ By the Orientation Lemma [17], G is not (k, ℓ)-sparse
    end if
    return $\textsc{SupersetSparsity}_{k,\ell}(D, \emptyset)$ ▷ Return `true` if there is no violating set $X \supsetneq \emptyset$
end procedure

Proposition 3. *Algorithm 3 runs in $O(T_{MF}(n) + T_{RC}(n, \ell))$ time.*

Proof. By Lemma 2, computing a k-indegree-bounded orientation takes $O(T_{MF}(n))$ time. Together with Proposition 1, which gives a running time of $O(T_{RC}(n, \ell))$ for $\textsc{SupersetSparsity}$, this yields the claimed bound. ■

2.4 The range $k < \ell < 2k$

For the range $k < \ell < 2k$, we invoke Lemma 1 in the special case $t = 1$, which gives an efficient test for whether there exists a violating vertex set containing a fixed vertex. Running this test for each vertex decides (k, ℓ)-sparsity, but it does not improve on quadratic time, so we need a faster approach. The crucial ingredient is the theorem of Nash–Williams [19], which implies that every (k, ℓ)-sparse graph with $k \leq \ell < 2k$ decomposes into k forests. We begin with an important definition.

Definition 2. Given a forest $F = (V, E)$, we say that F *saturates* a vertex set $X \subseteq V$ if the induced subgraph $F[X]$ is connected, i.e., it contains exactly $|X| - 1$ edges.

Lemma 4. *Let $k < \ell < 2k$ and suppose the edge set of $G = (V, E)$ decomposes into the edge sets of k spanning forests $F_1, \ldots, F_k$. Then any vertex set that violates the (k, ℓ)-sparsity of G is saturated by one of $F_1, \ldots, F_{\ell-k}$.*

Proof. Let X be a vertex set that is not saturated by any of $F_1, \ldots, F_{\ell-k}$. Then we have

$$i_G(X) = \sum_{i=1}^{k} i_{F_i}(X) \leq \sum_{i=1}^{\ell-k}(|X| - 2) + \sum_{i=\ell-k+1}^{k}(|X| - 1) = k|X| - 2(\ell - k) - (2k - \ell) = k|X| - \ell.$$

Hence X does not violate (k, ℓ)-sparsity. ■

Given a forest decomposition $F_1, \ldots, F_k$, the lemma above reduces our task to finding a violating vertex set that is saturated by one of the first $\ell - k$ forests. Observe that a forest saturates a set X if and only if one of its connected components does, so it suffices to check the components of $F_1, \ldots, F_{\ell-k}$. Accordingly, our first goal is to design a procedure that, for a given graph G and a spanning tree T of G, detects the presence in G of a violating vertex set saturated by T. We start with a simple observation.

Observation 1. Let $T = (V, F)$ be a tree and $X \subseteq V$ a vertex set saturated by T. Fix any $c \in V$, and let $T_1, \ldots, T_q$ be the connected components of $T \setminus \{c\}$. Then either $c \in X$, or there exists an index i such that $X \subseteq V(T_i)$ and X is saturated by T_i.

Proof. By definition, a vertex set X saturated by T induces a subtree in T. Consequently, either X contains c or $X \subseteq V(T_i)$ for some connected component T_i of $T \setminus \{c\}$. In the latter case, every edge of T induced by X also lies in T_i, hence X is saturated by T_i as well. ■

The observation above suggests a divide-and-conquer scheme via centroid decomposition [20]. Assume we have a k-indegree-bounded orientation of G, together with a spanning tree T. By Propositions 1 and 2, we can quickly test whether there exists a violating vertex set that contains the centroid c of T. If so, we are done; otherwise, we recurse on the connected components of T obtained by deleting c. The full implementation is given below.

Algorithm 4 Saturated Violation

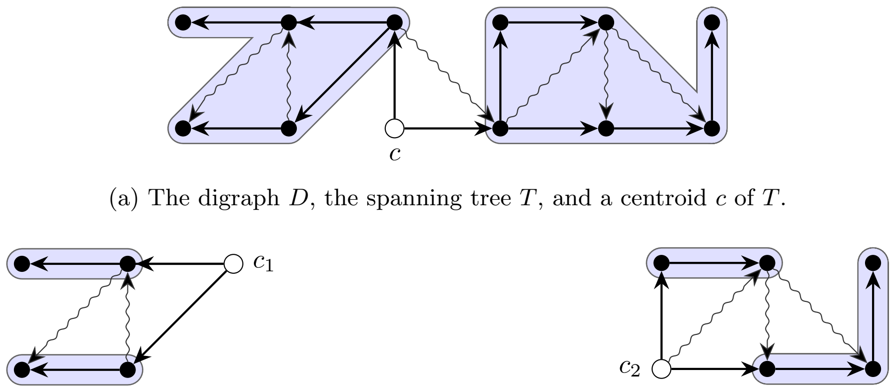
Input: A k-indegree-bounded digraph $D = (V, A)$, and a spanning tree $T = (V, F)$ of its underlying graph.
Output: `true` if there exists a vertex set that violates (k, ℓ)-sparsity in the underlying graph of D and is saturated by T; `false` if there is no violating vertex set in the graph.

procedure $\textsc{SaturatedViolation}_{k,\ell}(D = (V, A), T = (V, F))$
  $c \leftarrow$ the centroid of T
  $D_0 \leftarrow \textsc{Reorient}_{k,\ell}(D, \{c\})$ ▷ A k-indegree-bounded $\{c\}$-source orientation
  if $D_0 = \varnothing$ **or not** $\textsc{SupersetSparsity}_{k,\ell}(D_0, \{c\})$ **then** ▷ There is a violating set $X \supsetneq \{c\}$?
    return `true`
  end if
  $T_1, \ldots, T_q \leftarrow$ connected components of $(T \setminus \{c\})$
  for $i \leftarrow 1, \ldots, q$ **do**
    if $\textsc{SaturatedViolation}_{k,\ell}(D[V(T_i)], T_i)$ **then**
      return `true`
    end if
  end for
  return `false`
end procedure

Implementation details for SaturatedViolation**.** We find the centroid c using two depth-first searches: the first computes, for each vertex v, the sizes of the subtrees incident to v, and the second locates a vertex whose largest such subtree has size at most $n/2$. The components $T_1, \ldots, T_q$ with vertex sets $V_1, \ldots, V_q$ are obtained by q graph traversals. For each i, we build the induced digraph $D[V_i]$ by scanning the vertices of V_i, collecting their incoming arcs, and keeping only those whose tail also lies in V_i.

Note 6. If the underlying graph of D has a violating vertex set but none is saturated by T, then SaturatedViolation may return either `true` or `false`. This does not affect the correctness of the final recognition algorithm.

Example 2. Let $(k, \ell) = (2, 3)$, and consider the 2-indegree-bounded digraph D shown at the top of Figure 2, together with a spanning tree T whose edges are drawn as straight segments. First, find a centroid c of T. Since no violating vertex set contains c, SaturatedViolation recurses on the two subtrees of T incident to c and solves the corresponding subproblems independently.



(a) The digraph D, the spanning tree T, and a centroid c of T.

(b) The subgraph spanned by the left subtree of c, and a centroid c_1 of this subtree.

(c) The subgraph spanned by the right subtree of c, and a centroid c_2 of this subtree.

Figure 2: Two steps of the centroid decomposition.

For completeness, we record a simple fact that we will use repeatedly in the analysis.

Lemma 5. *Let $f(n) = n \cdot g(n)$ where $g : \mathbb{Z}_{>0} \to \mathbb{R}_{>0}$ is non-decreasing, and suppose $T(n) = O(f(n))$. Let q be a positive integer. Then for any positive integers $n_1, \ldots, n_q$ with sum N, it holds that*

$$\sum_{i=1}^{q} T(n_i) = O(f(N)).$$

Proof. For a sufficiently large positive constant C, the inequality $T(n) \leq C \cdot f(n)$ holds for all $n \in \mathbb{Z}_{>0}$. Then, using the monotonicity of g, we have

$$\sum_{i=1}^{q} T(n_i) \leq C \cdot \sum_{i=1}^{q} f(n_i) = C \cdot \sum_{i=1}^{q} n_i \cdot g(n_i) \leq C \cdot \sum_{i=1}^{q} n_i \cdot g(N) = C \cdot N \cdot g(N) = C \cdot f(N).$$

■

By the 'conquer' phase of Algorithm 4, we mean all steps excluding the recursive calls. We first analyze the running time of this phase, and then proceed to the analysis of the entire algorithm.

Proposition 4. *The conquer phase of Algorithm 4 runs in $O(T_{RC}(n, \ell - k))$ time.*

Proof. The two depth-first searches for finding a centroid take $O(n)$ time each. By Proposition 1, the calls to REORIENT and SUPERSETSPARSITY run in $O(n)$ and $O(T_{RC}(n, \ell - k))$ time, respectively. The subtrees $T_1, \ldots, T_q$ with vertex sets $V_1, \ldots, V_q$ are obtained using q graph traversals, the ith taking $O(|V_i|)$ time; hence, by Lemma 5, their total time is

$$O\left(\sum_{i=1}^{q} |V_i|\right) = O(n-1) = O(n).$$

Since D is k-indegree-bounded, each vertex of V_i contributes at most k in-arcs, so $D[V_i]$ is built in $O(|V_i|)$ time; applying Lemma 5 again gives a total time of $O(n)$. Combining these bounds, the overall running time of the conquer phase is

$$O(T_{RC}(n, \ell - k) + n) = O(T_{RC}(n, \ell - k)).$$

■

Proposition 5. *Algorithm 4 runs in $O(T'_{RC}(n, \ell - k) \log n)$ time.*

Proof. Consider all calls at a fixed recursion level: let the input graphs be $D_1, \ldots, D_p$, and set $n_i = |V(D_i)|$. By Proposition 4, the conquer step on D_i runs in time

$$O(T_{RC}(n_i, \ell - k)) \subseteq O(T'_{RC}(n_i, \ell - k)).$$

The digraphs D_i are pairwise vertex-disjoint subgraphs of the original digraph D, so the sum of n_i over all $i = 1, \ldots, p$ is at most n. Hence, by Lemma 5, the total time spent at this recursion level is $O(T'_{RC}(n, \ell - k))$. Choosing the centroid at line 2 ensures that each T_i has size at most $n/2$, so the recursion depth is $O(\log n)$. Therefore the overall running time of the algorithm is $O(T'_{RC}(n, \ell - k) \log n)$. ■

Let us return to the original problem, i.e., deciding whether a given graph G is (k, ℓ)-sparse. First, we compute a forest decomposition $F_1, \ldots, F_k$; then, for each connected component of $F_1, \ldots, F_{\ell-k}$, we use SATURATEDVIOLATION to detect the presence of a violating vertex set saturated by that component. The implementation is as follows.

Algorithm 5 Check Sparsity for $k < \ell < 2k$

Input: An undirected graph $G = (V, E)$.
Output: `true` if G is (k, ℓ)-sparse; `false` otherwise.

```
procedure CheckSparsity_{k,ℓ}(G = (V, E))
    F_1, ..., F_k ← a forest decomposition of G        ▷ Partition E into the edge sets of k spanning forests
    if no such decomposition exists then
        return false                                   ▷ By Nash-Williams' theorem [19], G is not (k, ℓ)-sparse
    end if
    D ← a k-indegree-bounded orientation of G          ▷ Use F_1, ..., F_k to construct D
    for i ← 1, ..., ℓ − k do
        T_{i,1}, ..., T_{i,q_i} ← connected components of F_i
        for j ← 1, ..., q_i do
            if SaturatedViolation_{k,ℓ}(D[V(T_{i,j})], T_{i,j}) then
                return false
            end if
        end for
    end for
    return true
end procedure
```

Implementation details for CHECKSPARSITY. Once a forest decomposition is known, a k-indegree-bounded orientation of G can be obtained in linear time: choose a root in each tree, and orient every edge from parent to child. With this orientation in hand, we find the components $T_{i,j}$ and build the induced subgraphs $D[V(T_{i,j})]$ exactly as in Algorithm 4.

Proposition 6. *Algorithm 5 runs in $O(T_{FD}(n, k) + T'_{RC}(n, \ell - k) \log n)$ time.*

Proof. Computing the forest decomposition takes $O(T_{FD}(n,k))$ time, after which the orientation D is obtained in linear time. Each connected component $T_{i,j}$ with $n_{i,j}$ vertices is identified in $O(n_{i,j})$ time by a single graph traversal. Applying Lemma 5 and noting that for a fixed i the sum of $n_{i,j}$ over $j = 1, \ldots, q_i$ is n, the total time over all components is

$$O((\ell - k) \cdot n) = O(n).$$

For each i, j, building the induced subgraph $D\,[V(T_{i,j})]$ takes $O(n_{i,j})$ time since each vertex contributes at most k arcs. By Proposition 5, the call to SaturatedViolation runs in $O(T'_{RC}(n_{i,j}, \ell - k)\log n_{i,j})$ time. Using Lemma 5 again and the fact that for a fixed i the sum of $n_{i,j}$ over all j is n, the total time of all calls is

$$O((\ell - k) \cdot T'_{RC}(n, \ell - k)\log n) = O(T'_{RC}(n, \ell - k)\log n).$$

Putting everything together, the overall running time is

$$O(T_{FD}(n,k) + n + T'_{RC}(n, \ell - k)\log n) = O(T_{FD}(n,k) + T'_{RC}(n, \ell - k)\log n).$$

■

2.5 The range $2k \le \ell < 3k$

In the range $2k \le \ell < 3k$, we follow the standard convention and require the sparsity bound only for vertex sets with at least 3 elements; throughout we also restrict attention to simple graphs. Adopting the pebble game paradigm, we construct a (k, ℓ)-sparse subgraph H of a given graph $G = (V, E)$ incrementally, starting from the empty graph. The edges of G are processed in an arbitrary but fixed order, and for each edge $uv \in E$ we test whether adding it preserves sparsity: the edge is inserted if and only if the resulting graph $H' = H \cup \{uv\}$ remains (k, ℓ)-sparse. Thus, the algorithm reduces to repeatedly checking whether the addition of a single edge violates the sparsity condition. The following observation gives an exact criterion for when such an insertion is feasible.

Observation 2. Let $2k \le \ell < 3k$, let $H = (V, E)$ be a (k, ℓ)-sparse graph, and let $u, v \in V$ be distinct with $uv \notin E$. Then the augmented graph $H' = (V, E \cup \{uv\})$ is (k, ℓ)-sparse if and only if no vertex set strictly containing $\{u, v\}$ violates the $(k, \ell + 1)$-sparsity of H.

Proof. By the (k, ℓ)-sparsity of H, every vertex set X that does not contain both u and v also satisfies the sparsity bound in H'. Consequently, any violating vertex set X in H' must contain $\{u, v\}$, and since $|X| \ge 3$, this inclusion is strict. For such sets, we have $i_{H'}(X) = i_H(X) + 1$, so X violates (k, ℓ)-sparsity in H' if and only if it violates $(k, \ell + 1)$-sparsity in H. ■

By the observation above and the $t = 2$ special case of Lemma 1, deciding whether H' is (k, ℓ)-sparse reduces to solving a rooted arc-connectivity problem — provided that we have a k-indegree-bounded orientation of H. As the implementation below shows, such an orientation can be maintained efficiently throughout the algorithm.

Algorithm 6 Check Sparsity for $2k \le \ell < 3k$

Input: An undirected graph $G = (V, E)$.
Output: `true` if G is (k, ℓ)-sparse; `false` otherwise.

1: **procedure** CheckSparsity$_{k,\ell}(G = (V, E))$
2: $\quad H \leftarrow (V, \emptyset)$ ▷ Initialize our sparse subgraph
3: $\quad D \leftarrow (V, \emptyset)$ ▷ Initialize our k-indegree-bounded orientation of H
4: $\quad$ **for** $uv \in E$ **do**
5: $\quad\quad D \leftarrow$ Reorient$_{k,\ell}(D, \{u, v\})$ ▷ A k-indegree-bounded $\{u, v\}$-source orientation
6: $\quad\quad$ **if** $D = \varnothing$ **or not** SupersetSparsity$_{k,\ell+1}(D, \{u, v\})$ **then** ▷ There is a violating set $X \supsetneq \{u, v\}$?
7: $\quad\quad\quad$ **return** `false`
8: $\quad\quad$ **end if**
9: $\quad\quad H \leftarrow H \cup \{uv\}$ ▷ Insert the edge uv into H
10: $\quad\quad D \leftarrow D \cup \{uv\}$ ▷ Insert an arc from u to v in D
11: $\quad$ **end for**
12: $\quad$ **return** `true`
13: **end procedure**

Note 7. After inserting the arc uv at line 10, the orientation D remains k-indegree-bounded. Indeed, only the indegree of v increases, and it was $0 < k$ before the insertion.

Proposition 7. *Algorithm 6 runs in $O(n \cdot T_{RC}(n, \ell + 1 - 2k))$ time.*

Proof. By Propositions 1 and 2, the calls to REORIENT and SUPERSETSPARSITY take $O(n)$ and $O(T_{RC}(n, \ell+1-2k))$ time, respectively. Since we process $m = O(n)$ edges in total, the overall running time is

$$O(n \cdot (T_{RC}(n, \ell + 1 - 2k) + n)) = O(n \cdot T_{RC}(n, \ell + 1 - 2k)).$$

∎

Note 8. It is possible to decide the sparsity of H' without using Lemma 1 by generalizing the ideas described in [14]. We handle an edge uv by checking, for each $w \in V \setminus \{u, v\}$, whether there exists a k-indegree-bounded orientation of H in which the sum of the indegrees of u, v and w is at most $3k - \ell - 1$. As in the pebble game, this can be carried out with augmenting paths in time $O(n^2)$. The resulting sparsity test is simpler than Algorithm 6, but its overall running time of $O(n^3)$ is asymptotically worse.

Note 9. With a minor modification, Algorithm 6 can be extended to find an inclusion-wise maximal (k, ℓ)-sparse subgraph of G. For a graph with n vertices and m edges, the running time is $O(m \cdot T_{RC}(n, \ell - 2k))$.

3 Summary and future work

Table 3 summarizes the best asymptotic running-time bounds we achieved for both optimization and recognition. For the recognition problem, more detailed bounds — given in terms of the running-time notation introduced in Section 2.1 — are presented in Table 2.

Table 3: Asymptotic running-time bounds for the main problems.

Range	Problem	Previous bounds	New bounds
$0 \le \ell \le k$	recognition	$O(n\sqrt{n \log n})$ [12]	$O(n^{1+o(1)})^*$
$k < \ell < 2k$	recognition	$O(n^2)^\dagger$ [13]	$O(n^{1+o(1)})^*$
	optimization	$O(nm)$ [13]	$O(n^2 + m)$
$2k \le \ell < 3k$	recognition	$\begin{cases} O(n^2) & \text{if } \ell = 2k \text{ [14]} \\ O(n^3) & \text{if } \ell > 2k \text{ [14]} \end{cases}$	$\begin{cases} O(n^2) & \text{if } \ell \le 2k + 1 \\ O(n^2 \log n) & \text{if } \ell > 2k + 1 \end{cases}$

* Under purely combinatorial implementations of our primitives, the new bounds become $O(n\sqrt{n})$ for the $\ell \le k$ recognition case and $O(n\sqrt{n \log n})$ for the $k < \ell < 2k$ recognition case.

† In the special case of Laman graphs, i.e., for $(k, \ell) = (2, 3)$ with $m = 2n - 3$, an $O(n^{1+o(1)})$-time recognition algorithm was developed in [15]. For the planar Laman case, a more specialized algorithm was proposed in [16] with a time complexity of $O(n \log^3 n)$.

Our paper on a quadratic-time algorithm for the maximum-weight (k, ℓ)-sparse subgraph problem is already available [21]. The paper on the recognition problem is currently in preparation and will be published soon.

References

[1] T. Jordán and A. Mihálykó. Minimum cost globally rigid subgraphs. In *Building Bridges II: Mathematics of László Lovász*, pages 257–278. Springer, 2020.

[2] A. Mihálykó. *Augmentation problems in count matroids and globally rigid graphs.* PhD thesis, Eötvös Loránd University, Budapest, 2022.

[3] D. Avis, N. Katoh, M. Ohsaki, I. Streinu, and S. Tanigawa. Enumerating constrained non-crossing minimally rigid frameworks. *Discrete & Computational Geometry*, 40:31–46, 2008.

[4] A. St. John. Kinematic joint recognition in CAD constraint systems. In *CCCG*, pages 173–178, 2012.

[5] J. Van Den Brand, L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. P. Gutenberg, S. Sachdeva, and A. Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514. IEEE, 2023.

[6] P. Arkhipov and V. Kolmogorov. A faster algorithm for the k-forest problem: breaking the $O_k(n^{3/2})$ complexity barrier. *arXiv preprint arXiv:2409.20314*, 2024.

[7] R. E. Tarjan. Edge-disjoint spanning trees, dominators, and depth-first search. Technical report, 1974.

[8] S. Alstrup, D. Harel, P. W. Lauridsen, and M. Thorup. Dominators in linear time. *SIAM Journal on Computing*, 28(6):2117–2132, 1999.

[9] H. N. Gabow. A matroid approach to finding edge connectivity and packing arborescences. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 112–122, 1991.

[10] E. A. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. In *Soviet Math. Doklady*, volume 11, pages 1277–1280, 1970.

[11] S. Even and R. E. Tarjan. Network flow and testing graph connectivity. *SIAM Journal on Computing*, 4(4):507–518, 1975.

[12] H. Gabow and H. Westermann. Forests, frames, and games: algorithms for matroid sums and applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 407–421, 1988.

[13] A. Lee and I. Streinu. Pebble game algorithms and sparse graphs. *Discrete Mathematics*, 308(8):1425–1437, 2008.

[14] P. Madarasi and L. Matúz. Pebble game algorithms and their implementations. In *12th Japanese-Hungarian Symposium on Discrete Mathematics and Its Applications March 21-24, 2023 in Budapest, Hungary*, 2023.

[15] O. Daescu and A. Kurdia. Towards an optimal algorithm for recognizing Laman graphs. In *2009 42nd Hawaii International Conference on System Sciences*, pages 1–10. IEEE, 2009.

[16] J. Rollin, L. Schlipf, and A. Schulz. Recognizing planar Laman graphs. In *27th Annual European Symposium on Algorithms (ESA 2019)*, pages 79–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019.

[17] S. L. Hakimi. On the degrees of the vertices of a directed graph. *Journal of the Franklin Institute*, 279(4):290–308, 1965.

[18] A. Frank. *Connections in combinatorial optimization*, volume 38. Oxford University Press Oxford, 2011.

[19] C. St. J. A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 1(1):445–450, 1961.

[20] C. Jordan. Sur les assemblages de lignes. *Journal fur die Reine und Angewandte Mathematik*, pages 185–190, 1869.

[21] B. Deák and P. Madarasi. Quadratic-time algorithm for the maximum-weight (k, ℓ)-sparse subgraph problem. *arXiv preprint arXiv:2511.20882*, 2025.