Parameterized Complexity of the Sparsest k-Subgraph Problem in Chordal Graphs [⋆]

Marin Bougeret, Nicolas Bousquet, Rodolphe Giroudeau, and Rémi Watrigant

LIRMM, Université Montpellier 2, France

Abstract. In this paper we study the Sparsest k-Subgraph problem which consists in finding a subset of k vertices in a graph which induces the minimum number of edges. The Sparsest k-Subgraph problem is a natural generalization of the Independent Set problem, and thus is $\mathcal{NP}$-hard (and even $W[1]$-hard) in general graphs. In this paper we investigate the parameterized complexity of both Sparsest k-Subgraph and Densest k-Subgraph in chordal graphs. We first provide simple proofs that Densest k-Subgraph in chordal graphs is FPT and does not admit a polynomial kernel unless $\mathcal{NP} \subseteq co\mathcal{NP}/poly$ (both parameterized by k). More involved proofs will ensure the same behavior for Sparsest k-Subgraph in the same graph class.

1 Introduction

1.1 Presentation of the Problem and Related Work

Given a simple undirected graph $G = (V, E)$ and an integer k, the Sparsest k-Subgraph problem asks to find k vertices in G inducing the minimum number of edges. The decision version asks if there exists a k-subgraph inducing at most C edges. As a generalization of the classical independent set problem (for $C = 0$), Sparsest k-Subgraph is $\mathcal{NP}$-hard in general graphs, as well as $W[1]$-hard when parameterized by k (as independent set is $W[1]$-hard [10]). In addition, there is an obvious XP algorithm for Sparsest k-Subgraph when parameterized by k, as all subsets of size k can be enumerated in $\mathcal{O}(n^k)$ time, where n is the number of vertices in the graph.

Several problems closely related to Sparsest k-Subgraph have been extensively studied in the past decades. Among them, one can mention the maximization version of Sparsest k-Subgraph, namely the Densest k-Subgraph, for which several results have been obtained in general or restricted graphs. In [8], the authors showed that Densest k-Subgraph remains $\mathcal{NP}$-hard in bipartite, comparability and chordal graphs, and is polynomial-time solvable in trees, cographs, and split graphs. The complexity status of Densest k-Subgraph in interval graphs, proper interval graphs and planar graphs is left as an open problem, and is still not answered yet. More recently, [5] improved some of these results by showing that both Densest k-Subgraph and Sparsest k-Subgraph are polynomial-time solvable in bounded cliquewidth graphs, and [3] developed exact algorithms for Sparsest k-Subgraph, Densest k-Subgraph and other similar problems in general graphs parameterized by k and the maximum degree $\varDelta$ of the graph.

[⋆] This work has been funded by grant ANR **2010** BLAN **021902**

During the past two decades, a large amount of work has been dedicated to the approximability of Densest k-Subgraph in general graphs. So far, the best approximation ratio is $O(n^{\delta})$ for some $\delta < 1/3$ [9], while the only negative result is due to Khot [14] ruling out a PTAS under some complexity assumptions. Still concerning Densest k-Subgraph but in restricted graph classes, [16] developed a PTAS in interval graphs, and [7, 15] developed constant approximation algorithms in chordal graphs. In [17], we recently proved that Sparsest k-Subgraph remains $\mathcal{NP}$-hard in chordal graphs and admits a 2-appoximation algorithm.
We can also mention the dual version of Sparsest k-Subgraph, namely the maximum partial vertex cover problem, for which we are looking for k vertices in the input graph which *cover* the maximum number of edges. Very recently [1] and [13] independently proved the $\mathcal{NP}$-hardness of maximum partial vertex cover in bipartite graphs, which directly transfers to Sparsest k-Subgraph (since finding k vertices covering the maximum number of edges is equivalent to find $(n - k)$ vertices inducing the minimum number of edges).
More generally, Sparsest k-Subgraph, Densest k-Subgraph and maximum partial vertex cover fall into the family of *cardinality constrained optimization problems* introduced by Cai [6]. In its survey, the author proved that these three problems are $W[1]$-hard in regular graphs, and gives an XP algorithm for general graphs with a better running time than the trivial algorithm.
As said previously, Sparsest k-Subgraph and Densest k-Subgraph are natural generalizations of k-independent set and k-clique, and are thus important both from a theoretical and practical point of view. Our motivation is to study their computational (parameterized) complexity in graph classes where they remains $\mathcal{NP}$-hard whereas k-independent set and k-clique are polynomial-time solvable, such as the well-known class of perfect graphs and some of its subclasses. To that end, we study their parameterized complexity in the class of chordal graphs, an important subclass of perfect graphs which arises in many practical situations [12]. More precisely, we prove that both Sparsest k-Subgraph and Densest k-Subgraph in chordal graphs are fixed-parameter tractable and do not admit a polynomial kernel under some classical complexity assumptions. As we will see, the results are quite easy to obtain for Densest k-Subgraph, but require some efforts for Sparsest k-Subgraph.

1.2 Organization of the Paper

In the following section (Section 2), we recall the classical definitions of parameterized complexity and chordal graphs. Our two main results, namely the FPT algorithm and kernel lower bound for Sparsest k-Subgraph in chordal graphs, are presented respectively in Sections 4 and 5. Before all these, we study as an appetizer the parameterized complexity of Densest k-Subgraph in chordal graphs in Section 3. Due to space restrictions, some proofs and figures were omitted and placed in appendices.

2 Parameterized Algorithms, Chordal Graphs

Parameterized algorithms. An interesting way to tackle $\mathcal{NP}$-hard problems is the parameterized complexity. A parameterized problem Q is a subset of $\Sigma^* \times \mathbb{N}$, where the second component is called the *parameter* of the instance. A *fixed-parameter tractable* (FPT for short) problem is a problem for which there exists an algorithm which, given $(x, k) \in \Sigma^* \times \mathbb{N}$, decides whether $(x, k) \in Q$ in time $f(k)|x|^{O(1)}$ for some computable function f. Such an algorithm becomes efficient with an hopefully small parameter. A *kernel* is a polynomial algorithm which, given $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs an instance (x', k') such that $(x, k) \in Q \Leftrightarrow (x', k') \in Q$ and $|x'| + k' \leq f(k)$ for some computable function f. The existence of a kernel is equivalent to the existence of an FPT-algorithm. Nevertheless one can ask the function f to be a polynomial. If so, then the kernel is called a *polynomial kernel.* If a problem admits a polynomial kernel, then it roughly means that we can, in polynomial time, compress the initial instance into an instance of size $poly(k)$ which contains all the hardness of the instance. In order to rule out polynomial kernels, we will use the recent concept of cross-composition [2].

Roughly speaking, a cross-composition is a polynomial reduction from t instances of a (non-parameterized) problem A to a single instance of a parameterized problem B such that the constructed instance is positive iff one of the input instances is positive. In addition, the parameter of the constructed instance must be of size polynomial in the maximum size of the input instances and the logarithm of t. It is known that if A is $\mathcal{NP}$-hard and A cross-composes into B, then B cannot admit a polynomial kernel under some complexity assumptions. For a stronger background concerning the parameterized complexity, we refer the reader to [10]. Formal definitions of cross-compositions and related notions are given in Appendix C.

Chordal graphs. A graph $G = (V, E)$ is a chordal graph if it does not contain an induced cycle of length at least four. As said previously, chordal graphs form an important subclass of perfect graphs. One can also equivalently define chordal graphs in terms of a special tree decomposition. Indeed, it is known [11] that a graph $G = (V, E)$ is a chordal graph if and only if one can find a tree $T = (\mathcal{X}, A)$ with $\mathcal{X} \subseteq 2^V$ such that for all $v \in V$, the set of nodes of T containing v, that is $\mathcal{X}_v = \{X \in \mathcal{X} : v \in X\}$, induces a (connected) tree, and such that for all $u, v \in V$ we have $\{u, v\} \in E$ if and only if $\mathcal{X}_u \cap \mathcal{X}_v \neq \emptyset$. Moreover, given a chordal graph, this corresponding tree can be found in polynomial time. From this definition, it is clear that each $X \in \mathcal{X}$ induces a clique in G.

3 Appetizer: Parameterized Complexity of Densest k-Subgraph in Chordal Graphs

The goal of this section is to prove the following result:

Theorem 1. Densest k-Subgraph *in chordal graphs is FPT and does not admit a polynomial kernel unless $\mathcal{NP} \subseteq co\mathcal{NP}/poly$, both parameterized by k.*

Proof. First, concerning the fixed-parameter tractability, notice that if the input graph G contains a clique of size k or more (which can be tested in polynomial time in chordal graphs), then it must be an optimal solution. Otherwise, it implies that the treewidth of G is upper bounded by k (since the treewidth equals the maximum clique number in chordal graphs), and we can apply the dynamic programming of

[4] over a classical tree decomposition of G in order to compute an optimal solution in FPT time.

Then, concerning the kernel lower bound, let us show that Densest k-Subgraph cross-composes into itself (parameterized by k). Indeed, suppose that we are given a sequence of t chordal graphs on n vertices together with $k, C \in \mathbb{N}$ (i.e. t instances of Densest k-Subgraph in chordal graphs sharing the same number of vertices and values for k and C, which is the polynomial equivalence relation). Let us consider the disjoint union of these t graphs, and for all $i \in \{1, ..., t\}$, add a clique K_i on n^2 vertices connected to all vertices of G_i (and not connected to the others). One can easily prove that the resulting graph is a chordal graph on $t \cdot (n + n^2)$ vertices (recall that G_i is a chordal graph for all $i \in \{1, ..., t\}$, and we cannot create any cycle of length four or more with the clique K_i). Then, we prove that there is a set of size $(n^2 + k)$ vertices in the resulting graph inducing at least $(\binom{n^2}{2} + kn^2 + C)$ edges if and only if there exists $i \in \{1, ..., t\}$ such that G_i contains k vertices inducing C edges or more. First, one can easily verify that if G_i contains a subset X of size k inducing C edges at least, then $X \cup K_i$ induces the desired number of edges. On the contrary, one can prove that a whole clique K_i must be taken in the solution to induce the desired number of edges. Since the remaining vertices must have average degree at least n^2, there is no vertex in other components. Hence, since Densest k-Subgraph in chordal graphs is $\mathcal{NP}$-hard, it does not admit a polynomial kernel unless $\mathcal{NP} \subseteq co\mathcal{NP}/poly$. $\square$

An interesting fact about the previous cross-composition is that it also holds for interval graphs as long as the input graphs are interval graphs. Unfortunately, since the complexity ($\mathcal{NP}$-hard *versus* Polynomial) of Densest k-Subgraph in interval graphs is still unknown, we cannot conclude any negative result under the classical complexity assumptions. However, it still shows that the $\mathcal{NP}$-hardness of Densest k-Subgraph in interval graphs would imply that it does not even admit a polynomial kernel (unless $\mathcal{NP} \subseteq co\mathcal{NP}/poly$). Taking the contrapositive, and still under the same complexity assumption, we have that it is sufficient[1] to derive a polynomial kernel in order to show that Densest k-Subgraph in interval graphs is not $\mathcal{NP}$-hard.

4 FPT Algorithm for Sparsest *k*-Subgraph in Chordal Graphs

Definitions and Notations. Let $G = (V, E)$ be a chordal graph and $T = (\mathcal{X}, A)$ be its corresponding tree decomposition as defined in section 2. Recall that for each $X \in \mathcal{X}$, X induces a clique in G. Actually, T can be chosen such that each $X \in \mathcal{X}$ induces a maximal clique [11]. However, we will not make any supposition in the following, since the algorithm will modify the graph through its tree decomposition, and we will sometimes loose this maximality property.

We denote respectively by $\mathcal{L}$ and $\mathcal{I}$ the set of leaves and internal nodes of T (we have $\mathcal{X} = \mathcal{L} \cup \mathcal{I}$). In the following we suppose that T is rooted at an arbitrary

[1] A polynomial kernel is theoretically easier to find than a polynomial algorithm, since a polynomial (and even a constant size) kernel can be easily derived from a polynomial algorithm.

node X_r. Let $X \in \mathcal{X}$, we denote by $pred(X)$ the unique predecessor of X in T (by convention $pred(X_r) = \emptyset$), and by $succ(X)$ the set of successors of X in T. For a vertex $v \in V$ (resp. a node $X \in \mathcal{X}$), we denote by $d(v)$ (resp. $d(X)$) its degree in G (resp. in T). For a set of vertices $U \subseteq V$ (resp. set of nodes $A \subseteq \mathcal{X}$), we denote by $G[U]$ (resp. $T[A]$) the subgraph of G induced by U (resp. the subforest of T induced by A). We say that a vertex $v \in V$ is a *lonely*[2] vertex (*resp. almost lonely* vertex) if $|\mathcal{X}_v| = 1$ (resp. $|\mathcal{X}_v| = 2$), *i.e.* if it appears in only one (resp. two) nodes of T.

First Observations. A maximum independent set can be computed in polynomial time in chordal graphs (since chordal graphs are perfect). Hence, we first determine if there exists an independent set of size k. In this case, we return this set which is naturally an optimal solution. Thus, we assume in the following that the graph G does not contain an independent set of size k.

Notice that we can assume that for every leaf L of the tree we do not have $L \subseteq pred(L)$ (otherwise we can contract the two nodes). Therefore, for each leaf L of the tree, there is a vertex $x \in L$ such that $x \notin pred(L)$, i.e. x is a lonely vertex. Since there is no independent set of size k in G, and since lonely vertices of leaves are pairwise non adjacent, we have the following:

Observation 1 *We can assume that $|\mathcal{L}| < k$.*

Let us now state a simple property verified by optimal solutions. Let S be a set of k vertices. Assume that there are vertices $x \in S$ and $y \in V \setminus S$ such that $\mathcal{X}_y \subsetneq \mathcal{X}_x$. Then it means that $N(y) \subseteq N(x)$. Thus, if we replace x by y in the solution, the number of edges in the solution cannot increase. A set S is *closed under inclusion* if there is no vertex x in S such that there exists $y \in V \setminus S$ such that $\mathcal{X}_y \subsetneq \mathcal{X}_x$. So there always exists an optimal solution closed under inclusion.

Idea of the Algorithm. Our goal is to find an optimal solution closed under inclusion. First note that any optimal solution closed under inclusion must contain a lonely vertex per leaf of T. Indeed, as each leaf L is not included in $pred(L)$, there exists a lonely vertex x in L. Thus, either the solution intersects L, and since it is closed by inclusion it contains a lonely vertex, or we can take a vertex of the solution and replace it by x, which does not create any additional edge (since no vertex of $N(x) = L \setminus \{x\}$ was in the solution).

Our method can be summarized as follows. First, we take a lonely vertex in each leaf and guess a binary flag $w(L) \in \{0, 1\}$ for each leaf L which indicates whether another vertex of L has to (with value 1) or does not have to (with value 0) be taken in the solution. The width of such a branching is bounded according to Observation 1. Then, given a leaf L with $w(L) = 1$, we first try to add to the solution the "most interesting" vertex of the leaf (for example a lonely vertex). When this is not possible (the neighborhood of the vertices of L can be incomparable if these vertices appear on incomparable subtrees), we apply some branching rules that re-structure the tree and create new "interesting vertices".

[2] Notice that every lonely vertex is a so-called "simplicial vertex" (a vertex whose neighborhood is a clique). However, if a node of the tree is contained in another node, a simplicial vertex may not be a lonely one. Since we do not make any supposition on the tree T (we will in particular duplicate nodes during the algorithm), we will prefer the term "lonely".

Terminology for the Algorithm. The algorithm is a branching algorithm composed of pre-processing rules (which do not require branching) and branching rules. When a rule is applied, we assume that previous rules cannot be applied.

During the algorithm, a partial solution S (initialized to $\emptyset$) will be constructed, and the input graph $G = (V, E)$ together with k, T (and thus $\mathcal{X}$, $\mathcal{L}$ and $\mathcal{I}$) will be modified. To avoid heavy notation we will keep these variables to denote the current input, and denote by G_0 the original graph, and by N_0 the neighborhood function of G_0.

In the following, *taking* a vertex $v \in V$ in the solution means that v is added to S, and v is removed both from the graph G and the tree T (removing each of its occurrences). *Deleting* a vertex means removing the vertex from G and from T. If a leaf of T becomes empty after taking or deleting a vertex, then simply remove the leaf.

Let $F \in \mathcal{I}$ be a leaf of $T[\mathcal{I}]$ (*i.e.* a node of T which all successors are leaves). The node F is a *bad father* if there exists a vertex u which appears in at least two leaves of $succ(F)$. So a node is a bad father if the leaves attached to it are not vertex disjoint. We denote by $\#BF$ the number of bad fathers of the tree. Finally, we denote by $\#AL$ (for "almost leaf") the number of internal nodes of T whose at least one successor is a leaf. Notice that $\#AL, \#BF \leq |\mathcal{L}|$.

In addition, as said previously, we will put "flags" on some leaves $\mathcal{L}^* \subseteq \mathcal{L}$ by introducing a boolean function $w : \mathcal{L}^* \to \{0, 1\}$, which indicates whether it intersects the solution (value 1) or not (value 0). At the beginning of the algorithm we have $\mathcal{L}^* = \emptyset$. For a solution $S \subseteq V_0$, we say that S *respects* the flags w if for all $L \in \mathcal{L}^*$, $w(L) = 0$ iff $S \cap L = \emptyset$. During the algorithm we will use the term "guessing" the value $w(L)$ of $L \in \mathcal{L}$. By this, we mean that we try the two possible choices (consistent with the previous ones), creating at most two distinct executions of the algorithm. Notice that $\mathcal{L}^*$ will be implicitly updated (*i.e.* L belongs to $\mathcal{L}^*$ in the next executions if we have guessed $w(L)$).
We also add a function $g : \mathcal{L}^* \to 2^S$. Roughly speaking, we will modify g during the algorithm such that g remembers the neighbors of the remaining vertices V in the partial solution S already constructed. Notice that we introduced g only for the analysis, and more precisely for maintaining our invariants (see bellow).

Correctness and Time Complexity. As usually in a branching algorithm, we bound the time complexity by bounding both the depth and the maximum degree of the search tree. More precisely, we will show that:

- Each rule can be applied in FPT time.
- The branching degree of each branching rule is a function of k.
- Any branching rule strictly decreases $(k, \#AL, \#BF)$ using the lexicographic order, whose initial value only depends on the initial value of k (by Observation 1).
- Any pre-processing rule does not increase $(k, \#AL, \#BF)$ and decreases $|V| + |\mathcal{I}|$.

Thus, the number of branching steps of the search tree is a function of k only, whereas the number of steps between two branchings is polynomial in n (recall that $|\mathcal{X}|$ is polynomial in n), which leads to an FPT running time.

Recall that S denotes the partial current solution. Concerning the correctness of the algorithm, we will say that a rule is *safe* if it preserves all the following invariants:

1. The tree T is still a tree decomposition (as defined in 2) of G, which is an induced subgraph of G_0.
2. If a vertex of the partial solution is adjacent to a "surviving" vertex $v \in V$, then v must appear in a leaf where a flag is defined, *i.e.*: $N_0(S) \cap V \subseteq \bigcup_{L \in \mathcal{L}^*} L$.
3. The neighborhood of a surviving vertex u in the partial solution is defined by the union of $g(L)$ for each L in which u appears, *i.e.* $g : \mathcal{L}^* \to 2^S$ is such that $\forall u \in V$ we have $N_0(u) \cap S = \bigcup_{L \in \mathcal{X}_u \cap \mathcal{L}^*} g(L)$.
   In particular, this invariant implies that if there are $u, v \in V$ such that $\mathcal{X}_u \cap \mathcal{L}^* \subseteq \mathcal{X}_v \cap \mathcal{L}^*$ (*i.e.* v appears in at least as many labelled leaves as u), then we must have $N_0(u) \cap S \subseteq N_0(v) \cap S$ (*i.e.* v is adjacent to at least as many vertices of the solution as u).
4. If there is an optimal solution (closed under inclusion) $S^* \subseteq V$ such that $S \subseteq S^*$, and S^* respects the flags w, then one of the branching will output an optimal solution.

Reduction Rules. Notice that each of the following rules defines a new value for variables k, T, S, w and g. However, for the sake of readability we will not mention variables that are not modified. Due to space restrictions, all safeness proofs were placed in Appendix B.

Pre-Processing Rule 1: useless duplicated node.
If there exists $X \in \mathcal{X}$ such that $X \notin \mathcal{L}^*$ and $X \subseteq pred(X)$, then contract X and $pred(X)$ (*i.e.* delete X, and connect every $Y \in succ(X)$ to $pred(X)$).

Pre-Processing Rule 2: removing a (almost) lonely vertex.
If there exists $L \in \mathcal{L}^*$ such that $w(L) = 0$, then if L contains a lonely vertex v, delete v. Otherwise, if L contains an almost lonely vertex v, then delete v.

Branching Rule 1: taking a lonely vertex.
If there exists $L \in \mathcal{L}^*$ such that $w(L) = 1$ and L contains a lonely vertex v, then take v in the solution and decrease k by one. In addition, add v into $g(L)$, and if L does not become empty, then guess a new value $w(L)$.

Remark 1. At this point, since *Pre-Processing Rule 1* does not apply, it is clear that every leaf $L \in \mathcal{L} \setminus \mathcal{L}^*$ contains a lonely vertex. The following branching rule aims to process these leaves.

Branching Rule 2: processing leaves with no flag.
If there exists $L \in \mathcal{L} \setminus \mathcal{L}^*$, then take a lonely vertex $v \in L$ in the solution and decrease k by one. In addition, add v into $g(L)$, and if L does not become empty, guess a value $w(L)$.

Remark 2. At this point, notice that $\mathcal{L}^* = \mathcal{L}$, *i.e.* a flag has been assigned to each leaf. Indeed, suppose that there exists $L \in \mathcal{L} \setminus \mathcal{L}^*$. If L contains a lonely vertex, then *Branching Rule 1* must apply. Otherwise, *Pre-Processing Rule 1* must apply. In addition, there is no lonely vertex in the leaves, as otherwise *Branching Rule 1* or *Pre-Processing Rule 2* would apply.

Branching Rule 3: partitioning leaves of a bad father.
If there exists a bad father $F \in \mathcal{X}$, let $\mathcal{L}'$ be the set of leaves in $succ(F)$ and $C = \bigcup_{L \in \mathcal{L}'} L$ be the set of vertices contained these leaves. Partition C into the equivalence classes $C_1, ..., C_t$ of the following equivalence relation: two vertices $u, v \in C$ are equivalent if $\mathcal{X}_u \cap \mathcal{L}' = \mathcal{X}_v \cap \mathcal{L}'$ (*i.e.* u and v appear in the same subset of leaves of F). For all $i \in \{1, ..., t\}$, let $\mathcal{L}^i \subseteq \mathcal{L}'$ denote the subset of leaves in which vertices of C_i were before the partitioning. Then, replace the leaves of F by $C_1, ..., C_t$, and for all $i \in \{1, ..., t\}$, guess $w(C_i)$ and set $g(C_i) = \bigcup_{L \in \mathcal{L}^i} g(L)$.

Let us give the intuition of *Branching Rule 3*. This rule ensures that the set of leaves attached to a same node are vertex disjoint and that the partition was made in such a way that two vertices in the same leaf after the application of the rule were in the same subset of leaves before it. Notice that the remaining Branching Rules can create bad fathers, but decrease k.

Branching Rule 4: taking a lonely vertex in a father.
If there exists $L \in \mathcal{L}$ such that $pred(L)$ contains a lonely vertex v, then take v in the solution, delete k by one, and create a new leaf N adjacent to $pred(L)$ and containing vertices of $L \setminus \{v\}$. Finally, guess a value for $w(N)$ and set $g(N) = \{v\}$.

Branching Rule 5: taking an almost lonely vertex in a leaf.
If there exists $L \in \mathcal{L}$ such that $w(L) = 1$ and L contains an almost lonely vertex v (thus contained in L and $pred(L)$), then take v in the solution, decrease k by one, and create a new leaf N adjacent to $pred(L)$ and containing vertices of $pred(L) \setminus \{v\}$. If L does not become empty, then guess a new value $w(L)$. Finally, guess a value $w(N)$, add v into $g(L)$, and set $g(N) = \{v\}$.

End of the Algorithm.

Lemma 1. *If no rule can be applied then either G is empty or $k = 0$.*

According to the introduction and all the safeness lemmas, the size of the search tree is a function of k. Then, let us remark that all rules can be applied in FPT time. This is clear for *Pre-Processing Rules 1* and *2*, as well as for *Branching Rules 1, 2, 4* and *5*. Concerning *Branching Rule 3*, which consists in partitioning a subset of leaves, it runs in FPT time as long as $|\mathcal{L}|$ is a function of k. This is obviously the case at the beginning of the algorithm (since $|\mathcal{L}| < k$), and the number of leaves only increase by one in *Branching Rule 4* and *5*, and by a function of the previous number of leaves in *Branching Rule 3*. Since the branching rules are applied at most $f(k)$ times, we get the desired result.

Theorem 2. *There is an FPT algorithm for* SPARSEST k-SUBGRAPH *in chordal graphs, parameterized by k.*

However, the running time of the algorithm may be a tower of 2 of height k, since *Branching Rule 3* may create 2^t new leaves, where t is the number of previous leaves of the node F. Nevertheless, we can slightly modify the algorithm in order to obtain a $O^*(2^{k^2})$ running time[3]. Indeed, after the application of this rule, all leaves L such that $w(L) = 0$ can be gathered into one leaf, since all these vertices are not in the solution. And since all leaves are vertex disjoint, the number of leaves L such that $w(L) = 1$ is at most k (since one vertex of each leaf is in the solution). Hence, the number of leaves of F after the application of *Branching Rule 3* can actually be bounded by $k + 1$. Then, as said previously, the only other branching rules which increase the number of leaves are *Branching Rules 4* and *5*, which both add at most one leaf when they are applied. However, since these branching rules are decreasing k, the maximum number of leaves of a node F before the application of *Branching Rule 3* is $2k$. Hence, this rule (which upper bounds the running time of the algorithm) runs in time $O^*(2^{O(2k^2)})$ (we have at most 2^{2k} leaves and we choose at most k leaves such that $w(L) = 1$). For sake of readability, the presented algorithm does not contain this slight modification.

5 Kernel Lower Bound of Sparsest k-Subgraph in Chordal Graphs

Intuition of the proof. The following kernel lower bound is obtained using a cross-composition. It is an extension of our previous work [17], showing the $\mathcal{NP}$-hardness of Sparsest k-Subgraph in chordal graphs. Let us first give the intuition of this result, and then explain the modification we apply which leads to the kernel lower bound. We then explicit the whole construction of the cross-composition and give a formal proof of the result.

The $\mathcal{NP}$-hardness proof is a reduction from the classical k-clique problem in general graphs and roughly works as follows. Given an input instance $G = (V, E)$, $k \in \mathbb{N}$ of k-clique, we first build a clique A representing the vertices of G. We also represent each edge $e_j = \{u, v\} \in E$ by a gadget F_j, and connect the representative vertices of u and v in A to some vertices of F_j (see the left of Figure 1). The reduction will force the solution to take in A the representatives of $(n-k)$ vertices of G (corresponding to the complement of a solution S of size k in G), and also to take the same number of vertices among each gadget. The key idea is that the cost of a gadget F_j increases by one if it is adjacent to one of the selected vertices of A. Thus, since the goal is to minimize the cost, we will try to maximize the number of gadgets adjacent to the representatives of S (*i.e.* vertices we did not pick in A), the maximum being reached when S is a clique in G.

To adapt this reduction into a cross-composition, we add an *instance selector* composed of $2\log t$ gadgets adjacent to A (where t is the number of input instances of the cross-composition) which encodes the binary representation of each instance index. These gadgets have the same structure as the F_j. For technical reasons, this instance selector has to be duplicated many times, as well as the clique A which we must duplicate t times in order to encode the vertex set of each instance. The right of Figure 1 represents the construction in a simplified way. Let us now define formally the gadgets and state their properties.

Definition of a gadget. Let $T \in \mathbb{N}$ (we will set the value of T later). The vertex set of each gadget is composed of three sets of T vertices X, Y and Z, with

[3] The $O^*(.)$ notation avoids polynomial terms.

$X = \{x_1, ..., x_T\}, Y = \{y_1, ..., y_T\}$ and $Z = \{z_1, ..., z_T\}$. The set X induces an independent set, the set Z induces a clique, and there is a $(T-1)$-clique on $\{y_2, ..., y_T\}$. In addition, for all $i \in \{1, ..., T\}$, we connect y_i to all vertices of Z and to x_i. The left of Figure 1 summarizes the construction.
In the following cross-composition, we will force the solution to take $2T$ vertices among each gadget F. It is easy to see that the sparsest $2T$-subgraph of F is composed of the sets X and Z, which induces $\binom{T}{2}$ edges. In addition, if we forbid the set Z to be in the solution (if the gadget is adjacent to some picked vertices of A), then the remaining $2T$ vertices (namely X and Y) induce $(\binom{T}{2} + 1)$ edges.

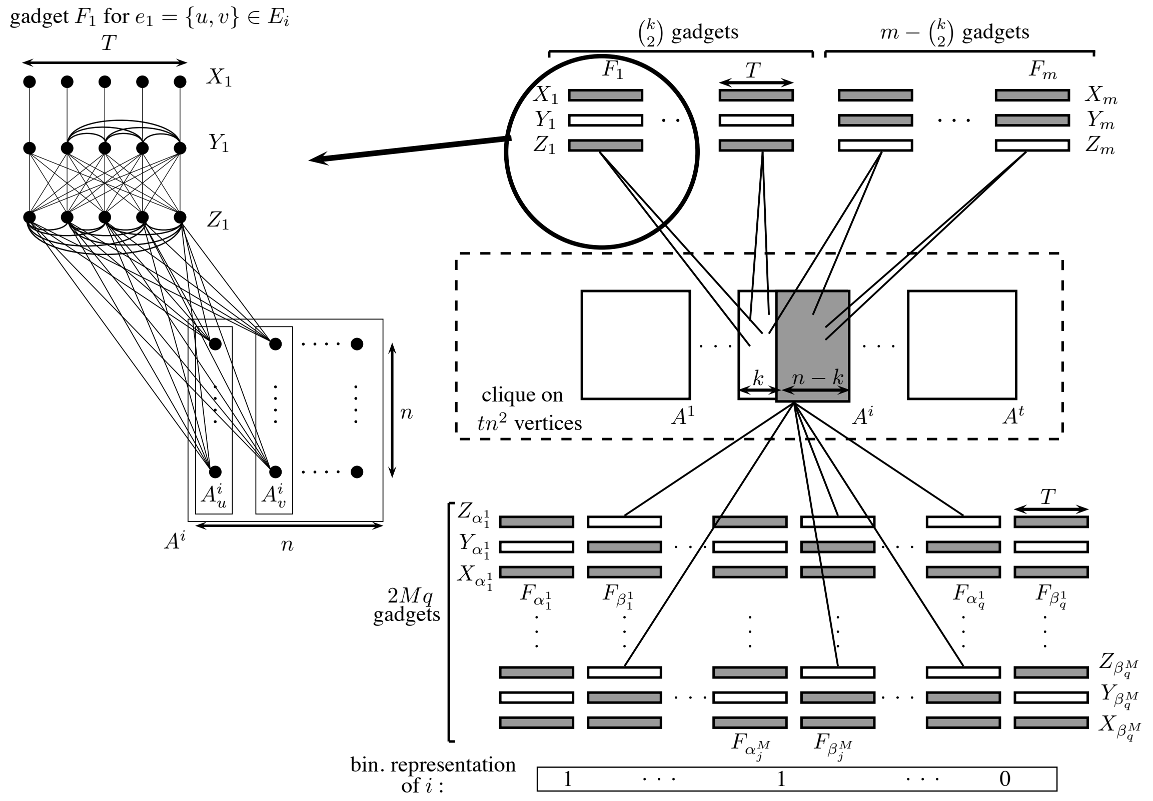

Fig. 1: Schema of the cross-composition (right) and a detailed gadget (left). Grey rectangles represent vertices of the solution, supposing that G_i contains a clique of size k. Notice that gadgets of the bottom have been drawn in the reverse direction (*e.g.* $X_{\beta_1^1}$ is below $Y_{\beta_1^1}$). Edges of the clique A have not been drawn for sake of clarity.

Theorem 3. Sparsest k-Subgraph *does not admit a polynomial kernel in chordal graphs unless* $\mathcal{NP} \subseteq co\mathcal{NP}/poly$ *(parameterized by* k*).*

Proof. Let $(G_1, k_1), ..., (G_t, k_t)$ be a sequence of t instances of k-clique, with $G_i = (V_i, E_i)$ for all $i \in \{1, ..., t\}$. W.l.o.g. we suppose that $t = 2^q$ for some $q \in \mathbb{N}$, and define $T = n(n-k)$ and $M = n^6$.

Our polynomial equivalence relation is the following: for $1 \leq i, j \leq t$, (G_i, k_i) is equivalent to (G_j, k_j) if $|V_i| = |V_j| = n$, $|E_i| = |E_j| = m$ and $k_i = k_j = k$. One can verify that this relation is a polynomial equivalence relation. In what follows we suppose that all instances of the sequence are in the same equivalence class. The output instance $G' = (V', E'), k', C'$ is defined as follows (see Figure 1):

– For each $i \in \{1, ..., t\}$ we construct a clique A^i on n^2 vertices, where A^i is composed of n subcliques $A^i_1, ..., A^i_n$. We also add all possible edges between all cliques $(A^i)_{i=1..n}$. Hence, $A = \bigcup_{i=1}^{t} A^i$ is a clique of size tn^2.
– Since all instances have the same number of edges, we construct m gadgets $(F_j)_{j=1..m}$, where each F_j is composed of X_j, Y_j and Z_j as described previously. For all $i \in \{1, ..., t\}$, if there is an edge $e_j = \{u, v\} \in E_i$, then we connect all vertices of Z_j to all vertices of A^i_u and A^i_v. Let us define $\mathcal{F} = \bigcup_{j=1}^{m} F_j$ the subgraph of all gadgets of the "edge selector".
– We add $2qM$ gadgets $(F_{\alpha_j^h})_{j=1..q}^{h=1..M}$ and $(F_{\beta_j^h})_{j=1..q}^{h=1..M}$, where all gadgets are isomorphic to the edge gadgets, and thus composed of $X_{\alpha_j^h}, Y_{\alpha_j^h}$ and $Z_{\alpha_j^h}$ (resp. $X_{\beta_j^h}, Y_{\beta_j^h}$ and $Z_{\beta_j^h}$) for all $h \in \{1, ..., M\}$ and all $j \in \{1, ..., q\}$. Let $i \in \{1, ..., t\}$, and consider its binary representation $b \in \{0, 1\}^q$. For all $j \in \{1, ..., q\}$, if the j^{th} bit of b equals 0, then connect all vertices of A^i to all vertices of $\bigcup_{h=1}^{M} Z_{\alpha_j^h}$. Otherwise, connect all vertices of A^i to all vertices of $\bigcup_{h=1}^{M} Z_{\beta_j^h}$. Let us define $\mathcal{B} = \bigcup_{h=1}^{M} \bigcup_{j=1}^{q} (F_{\alpha_j^h} \cup F_{\beta_j^h})$ the subgraph of all gadgets of the "instance selector".
– We set $k' = T + 2Tm + 4TqM$ and $C' = \binom{T}{2} + \binom{T}{2}(m + 2Mq) + (m - \binom{k}{2}) + Mq$.

It is clear that G', k' and C' can be constructed in time polynomial in $\sum_{i=1}^{t} |G_i| + k_i$. Then, one can verify that G' is a chordal graph. Indeed, it is known [12] that a graph is chordal if and only if one can repeatedly find a simplicial vertex (a vertex whose neighborhood is a clique) and delete it from the graph until it becomes empty. Such an ordering is called a simplicial elimination order. It is easily seen that for each gadget, X, Y and then Z is a simplicial elimination order (each gadget is only adjacent to the clique A *via* its set Z). Finally it remains the clique A which can be eliminated.

In addition, notice that the parameter k' is a polynomial in n, k and $\log t$ only and thus respect the definition of a cross-composition. We finally prove that there exists $i \in \{1, ..., t\}$ such that G_i contains a clique K of size k if and only if G' contains a set K' of k' vertices inducing C' edges or less.

Lemma 2. *If there exists $i \in \{1, ..., t\}$ such that G_i contains a k-clique, then G' contains k' vertices inducing at most C' edges.*

Proof. Suppose that $K \subseteq V_i$ is a clique of size k in G_i. W.l.o.g. suppose that $K = \{v_1, ..., v_k\}$, and that $\{\{u, v\}, u, v \in K\} = \{e_1, ..., e_{\binom{k}{2}}\}$. Let $b \in \{0, 1\}^q$ be the binary representation of i. We build K' as follows (see Figure 1).

– For all $j \in \{1, ..., \binom{k}{2}\}$, K' contains X_j and Z_j ($2T$ vertices inducing $\binom{T}{2}$ edges for each gadget F_j).
– For all $j \in \{\binom{k}{2} + 1, ..., m\}$, K' contains X_j and Y_j. ($2T$ vertices inducing $(\binom{T}{2} + 1)$ edges for each gadget F_j).

– For all $u \notin \{1, ..., k\}$, K' contains A^i_u (T vertices inducing $\binom{T}{2}$ edges).
– For all $h \in \{1, ..., M\}$, and all $j \in \{1, ..., q\}$, K' contains $X_{\alpha^h_j}$ and $X_{\beta^h_j}$. Moreover, if the j^{th} bit of b equals 1, then K' contains $Y_{\beta^h_j}$ and $Z_{\alpha^h_j}$, otherwise K' contains $Z_{\beta^j_j}$ and $Y_{\alpha^h_j}$ ($4T$ vertices inducing $(2\binom{T}{2} + 1)$ edges for each pair of gadgets $F_{\alpha^h_j}$ and $F_{\beta^h_j}$).

One can easily verify that K' is a set of k' vertices inducing C' edges. □

We terminate the proof by the following lemma, whose proof is in Appendix C:

Lemma 3. *If G' contains k' vertices inducing at most C' edges, then $\exists i \in \{1, ..., t\}$ such that G_i contains a k-clique.*

References

1. N. Apollonio and B. Simeone. The maximum vertex coverage problem on bipartite graphs. Discrete Applied Mathematics, (in press), 2013.
2. H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Cross-composition: A new technique for kernelization lower bounds. In STACS, pages 423–434, 2011.
3. E. Bonnet, B. Escoffier, V. Th. Paschos, and E. Tourniaire. Multi-parameter complexity analysis for constrained size graph problems: using greediness for parameterization. to appear in IPEC 2013.
4. N. Bourgeois, A. Giannakos, G. Lucarelli, I. Milis, and V. Th. Paschos. Exact and approximation algorithms for densest k-subgraph. In WALCOM, pages 114–125, 2013.
5. H. Broersma, P. A. Golovach, and V. Patel. Tight complexity bounds for FPT subgraph problems parameterized by clique-width. In Proceedings of the 6th international conference on Parameterized and Exact Computation, IPEC'11, pages 207–218, Berlin, Heidelberg, 2012. Springer-Verlag.
6. L. Cai. Parameterized complexity of cardinality constrained optimization problems. Computer Journal, 51(1):102–121, 2008.
7. D. Chen, R. Fleischer, and J. Li. Densest k-subgraph approximation on intersection graphs. In Proceedings of the 8th international conference on Approximation and online algorithms, pages 83–93. Springer, 2011.
8. D.G. Corneil and Y. Perl. Clustering and domination in perfect graphs. Discrete Applied Mathematics, 9(1):27 – 39, 1984.
9. U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. Algorithmica, 29:2001, 1999.
10. J. Flum and M. Grohe. Parameterized Complexity Theory. Springer, 2006.
11. F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. Journal of Combinatorial Theory, Series B, 16(1):47 – 56, 1974.
12. M. C. Golumbic. Algorithmic graph theory and perfect graphs. Academic Press, New York, USA, 1980.
13. G. Joret and A. Vetta. Reducing the rank of a matroid. CoRR, abs/1211.4853, 2012.
14. S. Khot. Ruling out ptas for graph min-bisection, dense k-subgraph, and bipartite clique. SIAM Journal of Computing, 36:1025–1071, 2004.
15. M. Liazi, I. Milis, and V. Zissimopoulos. A constant approximation algorithm for the densest k-subgraph problem on chordal graphs. Information Processing Letters, 108(1):29–32, 2008.
16. T. Nonner. PTAS for densest k-subgraph in interval graphs. In Proceedings of the 12th international conference on Algorithms and Data Structures, pages 631–641. Springer, 2011.
17. R. Watrigant, M. Bougeret, and R. Giroudeau. Approximating the sparsest k-subgraph in chordal graphs. to appear in WAOA 2013.

A Formal Definitions for Kernel Lower Bounds

In order to establish kernel lower bounds, we use the concept of cross-composition of [2]:

Definition 1 (Polynomial equivalence relation [2]). *An equivalence relation $\mathcal{R}$ on Σ^* is called a polynomial equivalence relation if the two following conditions hold:*

- *There is an algorithm that given two strings $x, y \in \Sigma^*$, decides whether x and y belong to the same equivalence class in $(|x| + |y|)^{O(1)}$ time.*
- *For any finite set $S \subseteq \Sigma^*$, the equivalence relation $\mathcal{R}$ partitions the elements of S into at most $(max_{x \in S}|x|)^{O(1)}$ classes.*

Definition 2 (OR-cross-composition [2]). *Let $L \subseteq \Sigma^*$ be a set and let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that L OR-cross-composes into Q if there is a polynomial equivalence relation $\mathcal{R}$ and an algorithm which, given t strings belonging to the same equivalence class of $\mathcal{R}$, computes an instance $(x^*, k^*) \in \Sigma^* \times \mathbb{N}$ in time polynomial in $\sum_{i=1}^{t} |x_i|$ such that:*

- *$(x^*, k^*) \in Q \Leftrightarrow x_i \in L$ for some $1 \leq i \leq t$*
- *k^* is bounded by a polynomial in $max_{i=1}^{t}|x_i| + \log t$*

Theorem 4 ([2]). *If some set $L \subseteq \Sigma^*$ is $\mathcal{NP}$-hard and L OR-cross-composes into the parameterized problem Q, then there is no polynomial kernel for Q unless $\mathcal{NP} \subseteq co\mathcal{NP}/poly$.*

B Missing Proofs of Section 4

B.1 Safeness of *Pre-Processing Rule 1*

Proof. The new tree still verifies invariant 1. As $X \notin \mathcal{L}^*$, $\bigcup_{L \in \mathcal{L}^*} L$ remains unchanged, and since S is also unchanged, invariant 2 is clearly preserved. In the same way, as $X \notin \mathcal{L}^*$, $\bigcup_{L \in \mathcal{X}_u \cap \mathcal{L}^*} g(L)$ remains unchanged for any u, and invariant 3 is preserved. Invariant 4 remains true as we do not modify S nor w.

Let us now check what is decreasing when applying this rule. Notice that this rule may increase $\#BF$ as $pred(X)$ may become a bad father. However, in this case the rule decreases $\#AL$, and thus $(k, \#AL, \#BF)$ decreases. Otherwise (if $\#BF$ does not increase), either $\#AL$ decreases, or $(k, \#AL, \#BF)$ remains unchanged and $|V| + |\mathcal{I}|$ decreases.

□

B.2 Safeness of *Pre-Processing Rule 2*

Proof. Here again invariant 1 still holds. Then, since we just remove a vertex from the graph and do not modify the solution, invariant 2 is still true. For the same reason, and since $g(L)$ is not modified either, invariant 3 holds. Let us prove that invariant 4 is preserved. Consider an optimal solution S^* closed under inclusion which satisfies $S \subseteq S^*$ and the flags on the leaves. As $w(L) = 0$, no vertex of L is used in S^*, and in particular $v \notin S^*$. Thus, vertices of $S^* \setminus S$ are still in the

remaining graph and the invariant still holds.
Then, obviously $|V| + |\mathcal{I}|$ decreases while k remains unchanged. The only case in which $\#BF$ may increase is when $L = \{v\}$ and $succ(pred(L)) = L$ (*i.e.* L was the unique leaf of $pred(L)$), and $pred(pred(L))$ is an almost leaf). In this case L is deleted and thus $pred(L)$ now becomes a leaf and $pred(pred(L))$ may become a bad father. However in this case $pred(L)$ and $pred(pred(L))$ were two almost leaves, and thus the deletion of v (and thus L) decreases $\#AL$, which proves that $(k, \#AL, \#BF)$ cannot increase. □

B.3 Safeness of *Branching Rule 1*

Proof. Here again a vertex is deleted from the graph and thus invariant 1 is still verified. In addition, neighbors of v in the remaining graph must appear in the leaf L only (since v is lonely), which receives a flag $w(L)$. Hence invariants 2 holds. Since v has been added to $g(L)$, invariant 3 holds too. For the last invariant, let us consider S^* an optimal solution closed under inclusion such that $S \subseteq S^*$ and S^* satisfies the flags of w. Suppose that $v \notin S^*$. Let $x \in L \cap S^*$ (such a vertex must exist, according to $w(L)$). Let us prove that $N_0(v) \cap S^* \subseteq N_0(x) \cap S^*$ (as this will imply that replacing x by v in S^* cannot increase its cost). By invariant 3, it holds that $N_0(v) \cap S \subseteq N_0(x) \cap S$. By invariant 1 and by definition of the tree T, it holds that $N_0(v) \cap S^* \cap V \subseteq N_0(x) \cap S^* \cap V$. Since $S^* = S \cup (S^* \cap V)$, the result follows and invariant 4 is true.
Finally, it is clear that k decreases. □

B.4 Safeness of *Branching Rule 2*

Proof. Using the same arguments as in *Branching Rule 1*, invariants 1, 2 and 3 hold. Then, let us consider S^* an optimal solution closed under inclusion such that $S \subseteq S^*$ and S^* satisfies the flags of w. Suppose that $v \notin S^*$. Since L has no flag, two cases may happen: either $S^* \cap L = \emptyset$ or $S^* \cap L \neq \emptyset$. In the first case, since invariant 2 implies $N_0(v) \cap S = \emptyset$, and since v is a lonely vertex, we have $N_0(v) \cap S^* = \emptyset$. Hence replacing any other vertex of S^* by v cannot increase its number of induced edges. Suppose now that $S^* \cap L \neq \emptyset$, and let $x \in L \cap S^*$. As in the proof of *Branching Rule 1*, let us prove that $N_0(v) \cap S^* \subseteq N_0(x) \cap S^*$ (as this will imply that replacing x by v in S^* cannot increase its cost). By invariant 3, it holds that $N_0(v) \cap S \subseteq N_0(x) \cap S$. By invariant 1 and by definition of the tree T, it holds that $N_0(v) \cap S^* \cap V \subseteq N_0(x) \cap S^* \cap V$. Since $S^* = S \cup (S^* \cap V)$, the result follows and invariant 4 is true.
Here again it is clear that k decreases.

B.5 Safeness of *Branching Rule 3*

Proof. Notice first that by construction $\#BF$ decreases, whereas k and $\#AL$ remain unchanged.

Let us now check the invariants. Since vertices which appear in a leaf before the transformation still appear on some leaves, invariant 2 is preserved. By Remark 2, no leaf contains a lonely vertex. Thus, all vertices of C are contained in F and thus induce a clique. Since we do not modify F, no vertex nor edge has been removed from the graph, and invariant 1 still holds. For proving that invariant 3 still holds, let $i \in \{1, ..., t\}$ and $u \in C_i$. Before the partitioning we had:

$$\begin{aligned} N_0(u) \cap S &= \bigcup_{L \in \mathcal{X}_u \cap \mathcal{L}} g(L) \\ &= \left(\bigcup_{L \in \mathcal{X}_u \cap \mathcal{L}'} g(L) \right) \cup \left(\bigcup_{L \in \mathcal{X}_u \cap (\mathcal{L} \setminus \mathcal{L}')} g(L) \right) \end{aligned}$$

And by definition, we now have $\bigcup_{L \in \mathcal{X}_u \cap \mathcal{L}'} g(L) = g(C_i)$. Hence, the invariant is preserved.
Let us now turn to invariant 4. Consider a solution S^* optimal and closed by inclusion satisfying $S \subseteq S^*$ and the flags w on the leaves. If we consider the branching where every new leaf C_i receives the right flag with respect to $S^* \cap C_i$, then the solution S^* satisfies the assigned flags, and invariant 2 holds. □

B.6 Safeness of *Branching Rule 4*

Proof. First, it is clear that invariant 1 still holds, since we just removed a vertex v from the graph, and duplicated a node of the tree in a leaf. Then, since we created a leaf N containing all neighbors of v, and since we assigned a value $w(N)$ for this new leaf, invariant 2 is preserved. Concerning invariant 3, notice that for all $u \in pred(L)$, its neighborhood in the partial solution after the branching rule ($N_0(u) \cap S$) is exactly the union of its neighborhood in the previous partial solution and $\{v\}$. By definition of $g(N)$, and since u now belongs to N, this proves that the invariant is still true.
Let us know prove that invariant 4 still holds. Let S^* be an optimal solution closed under inclusion which satisfies $S \subseteq S^*$ and the already assigned flags w. If $S^* \cap pred(L) \neq \emptyset$ then the result is straightforward since v is lonely. Otherwise, there are two cases:

- first case: there exists $L \in pred(L)$ such that $S^* \cap L \neq \emptyset$. In this case, let $u \in S^* \cap L$. Since L does not contain any lonely vertex (see Remark 2), S^* is actually not closed under inclusion, which proves that this case is impossible.
- second case: for all $L \in pred(L)$ we have $S^* \cap L = \emptyset$. In this case it means that $N_0(v) \cap S^* = \emptyset$ and thus we can replace any other vertex of S^* by v without increasing its cost.

Finally, it is clear that k decreases.

B.7 Safeness of *Branching Rule 5*

Proof. Since we just removed a vertex from G and duplicated a node, creating a leaf, invariant 1 still holds. In addition, neighbors of v in the remaining graph must appear in the new leaf N, which receives a flag $w(N)$. Hence invariant 2 and 3 also hold (notice that we added v into $g(L)$, and that $g(N)$ has been set to $\{v\}$). Concerning invariant 4, let S^* be an optimal solution closed under inclusion, such that $S \subseteq S^*$, and respecting the flags w. Let $x \in S^* \cap L$ (such a vertex must exist, according to $w(L)$), and suppose that $v \notin S^*$. By invariant 2 it holds that

$N_0(v) \cap S \subseteq N_0(x) \cap S$. Since there is no lonely vertex in L (cf Remark 2), it holds that $N_0(v) \cap S^* \cap V \subseteq N_0(v) \cap S^* \cap V$. Since $S^* = S \cup (S^* \cap V)$, this proves that invariant 4 is preserved.
Finally, k strictly decreases.

B.8 Proof of Lemma 1

Proof. Let us first prove that the depth of T is at most 1 (that is, T is a star). Suppose by contradiction that there exists an internal node F of depth at least 1, *i.e.* at least one leaf is adjacent to F, and $F \neq X_r$ (and thus $pred(F)$ exists). By *Pre-Processing Rule 2* and *Branching Rule 5*, no leaf of F has an almost lonely vertex. So every vertex which appears in F and a leaf of F also appears in $pred(F)$ (since otherwise *Branching Rule 3* would apply). In addition, *Pre-Processing Rule 1* ensures that $F \nsubseteq pred(F)$. Then there exists a vertex v in F which is not in $pred(F)$. Hence v must be a lonely vertex of F and *Branching Rule 4* can be applied, a contradiction.

So T is a star rooted on X_r. Since *Branching Rule 3* cannot be applied, leaves of X_r are vertex disjoint. So every vertex which appears in a leaf is a lonely or an almost lonely vertex. Let L be such a leaf. If $w(L) = 0$, then *Pre-Processing Rule 2* can be applied. Otherwise *Branching rule 1* or *5* can be applied as long as X_r has a leaf.

Hence G is now reduced to a clique. If $k = 0$ then we already have the solution and can output it. If $k > 0$, then since each vertex is a lonely one, *Branching Rule 1* can apply and we can thus choose arbitrarily any remaining vertex.

Thus, the algorithm ends when the graph is empty or when $k = 0$. If the graph is empty and $k > 0$, then we know that the current branching is not the right one, and then the output does not provide an optimal solution. In the other cases, we compare the costs of all produced solutions (in each branching). Since invariant 4 is preserved in all pre-processing and branching rules, one of the branch of the search tree must provide a solution of optimal cost. Therefore the minimum over all the possible branchings provides a solution with an optimal cost, which finishes the proof. □

C Missing Proofs of Section 5

C.1 Proof of Lemma 3

Let us first state some definitions.

Definitions. Suppose now that K' is a set of k' vertices inducing C' edges. For a set $S \subseteq V'$, we denote by $tr(S) = S \cap K'$ the trace of the solution on S. For all $v \in V'$, let $\mu(v) = |tr(N(v))|$ be the number of neighbors of v belonging to K'.
Let $\mathcal{I} = \{1, ..., m\} \cup \{\alpha_j^h\}_{j=1..q}^{h=1..M} \cup \{\beta_j^h\}_{j=1..q}^{h=1..M}$ be the set of all indices of gadgets. As in the definition of the gadgets given above, we define for all $\gamma \in \mathcal{I}$ the sets $X_\gamma = \{x_1^\gamma, ..., x_T^\gamma\}$, $Y_\gamma = \{y_1^\gamma, ..., y_T^\gamma\}$ and $Z_\gamma = \{z_1^\gamma, ..., z_T^\gamma\}$.
We define $E_0 = \{\gamma \in \mathcal{I}$ such that $\forall x \in tr(A)$, no vertex of Z_γ is adjacent to $x\}$,

i.e. E_0 represents the indices of all gadgets F_γ which are not adjacent to vertices of the solution among the clique A. Then, define $E_1 = \mathcal{I} \setminus E_0$, which represents indices of gadgets which are adjacent to at least one vertex of $tr(A)$.

In the three following lemmas (4, 5 and 6), we show that we can restructure the solution inside each gadget in order to encode a solution for the k-clique instance. To do so, we define the notion of *safe replacement*:

Safe replacements. Let $u \in K'$ and $v \in V' \backslash K'$. We say that $(K' \backslash \{u\}) \cup \{v\}$ is a safe replacement if we have $\mu(v) \leq \mu(u)$ if $\{u,v\} \notin E'$ and $\mu(v) - 1 \leq \mu(u)$ if $\{u,v\} \in E'$. It is easily seen that in this case $(K' \backslash \{u\}) \cup \{v\}$ does not induce more edges than K'. For the sake of readability, we will keep the same notations and update the set K' when applying replacements, as well as the sets E_0 and E_1 when replacing vertices of A (*e.g.* if there exists $\gamma \in E_1$ such that F_γ is adjacent to a unique vertex $u \in tr(A)$, and if a replacement removes u from the solution, then γ now belongs to E_0).

Lemma 4. *Without loss of generality (and optimality of K'), we can suppose that for all $\gamma \in \mathcal{I}$ we have $X_\gamma \subseteq K'$.*

Proof. Let $S = \bigcup_{\gamma \in \mathcal{I}} X_\gamma$. Since we have $k' > |S|$, we have $K' \backslash S \neq \emptyset$. Suppose that there exists $\gamma \in \mathcal{I}$ and $i \in \{1, ..., T\}$ such that $x_i^\gamma \notin K'$. Recall that y_i^γ is the only neighbor of x_i^γ. If $y_i^\gamma \notin K'$, then we have $\mu(x_i^\gamma) = 0$ and we can thus safely replace any other vertex of $K' \backslash S$ by x_i^γ. Now, if $y_i^\gamma \in K'$, then $\mu(x_i^\gamma) = 1$. Since x_i^γ and y_i^γ are adjacent, $(K' \backslash \{y_i^\gamma\}) \cup \{x_i^\gamma\}$ is a safe replacement. □

In the following, we suppose that for all $\gamma \in \mathcal{I}$ we have $X_\gamma \subseteq K'$.

Lemma 5. *K' can be safely modified such that one of the two following cases must happen (see Figure 2):*

- *case A1: for all $\gamma \in E_0$ we have $tr(Z_\gamma) = Z_\gamma$.*
- *case A2: for all $\gamma \in E_0$ we have $tr(Y_\gamma) = \emptyset$.*

Proof. Let us first restructure each gadget of E_0 separately. For all $\gamma \in E_0$ such that $tr(Y_\gamma) \neq \emptyset$ and $tr(Z_\gamma) \neq Z_\gamma$, let $j_0 = \max\{j \in \{1, ..., T\} : y_j^\gamma \in tr(Y_\gamma)\}$ and let j_1 be such that $z_{j_1}^\gamma \notin tr(Z_\gamma)$. Recall that Lemma 4 ensures that $x_{j_0}^\gamma$ is in K'. If $j_0 \neq 1$, then $\mu(y_{j_0}^\gamma) = y + z + 1$, where $y = |N(y_{j_0}^\gamma) \cap tr(Y_\gamma)|$ and $z = |N(y_{j_0}^\gamma) \cap tr(Z_\gamma)|$. On the other side, we have $\mu(z_{j_1}^\gamma) \leq y + z + 1$ (more precisely, $\mu(z_{j_1}^\gamma) = y + z + 1$ if $y_1^\gamma \in K'$, and $\mu(z_{j_1}^\gamma) = y + z$ if $y_1^\gamma \notin K'$). Roughly speaking, this switch ensures that we necessarily "loose" the edge due to the vertex of X^γ and we gain at most one edge due to y_1^γ. Hence $\mu(z_{j_1}^\gamma) \leq \mu(y_{j_0}^\gamma)$ and $(K' \backslash \{y_{j_0}^\gamma\}) \cup \{z_{j_1}^\gamma\}$ is a safe replacement. If $j_0 = 1$, then it means that $tr(Y_\gamma) = \{y_1^\gamma\}$. Suppose that there exists j_1 such that $z_{j_1}^\gamma \notin tr(Z_\gamma)$. We have $\mu(y_1^\gamma) = z + 1$ where $z = |N(y_1^\gamma) \cap tr(Z_\gamma)|$, and $\mu(z_{j_1}^\gamma) = z + 1$. Here again $(K' \backslash \{y_1^\gamma\}) \cup \{z_{j_1}^\gamma\}$ is a safe replacement. After all these replacements, given any $\gamma \in E_0$, $tr(Y_\gamma) \neq \emptyset$ implies that $tr(Z_\gamma) = Z_\gamma$.
Then, we proceed to replacements between gadgets F_γ, $\gamma \in E_0$. If one can find $a, b \in E_0$ such that $tr(Y_a) \neq \emptyset$ and $tr(Z_b) \neq Z_b$, then let j_0 be such that $y_{j_0}^a \in tr(Y_a)$ and let j_1 be such that $z_{j_1}^b \notin tr(Z_b)$. We have $\mu(y_{j_0}^a) \geq T + 1$ and $\mu(z_{j_1}^b) \leq T - 1$. Thus, $(K' \backslash \{y_{j_0}^a\}) \cup \{z_{j_1}^b\}$ is a safe replacement.

These replacements end either when $tr(Y_\gamma) = \emptyset$ for all $\gamma \in E_0$ or when $tr(Z_\gamma) = Z_\gamma$ for all $\gamma \in E_0$, which achieves the proof of Lemma 4. □

Lemma 6. *K' can be safely modified such that one of the two following cases must happen (see Figure 2):*

- *case B1: for all $\gamma \in E_1$ we have $tr(Y_\gamma) = Y_\gamma$.*
- *case B2: for all $\gamma \in E_1$ we have $tr(Z_\gamma) = \emptyset$.*

Proof. The proof is roughly based on the fact that replacing a vertex of Z_γ by a vertex of Y_γ permits to "loose" at least one edge with vertices A and "gain" one edge with a vertex of X_γ. Let us formally prove Lemma 6. Similarly to the proof of Lemma 5, we first restructure each gadget of E_1 separately: for all $\gamma \in E_1$ such that $tr(Z_\gamma) \neq \emptyset$ and $tr(Y_\gamma) \neq Y_\gamma$, let $j_0 = \max\{j \in \{1, ..., T\} : y_j^\gamma \notin K'\}$ and let j_1 be such that $z_{j_1}^\gamma \in tr(Z_\gamma)$. Recall that by definition of E_1, there exists $i, j \in \{1, ..., n\}$ such that $z_{j_1}^\gamma$ is adjacent to a_i^j. We have $\mu(z_{j_1}^\gamma) \geq y + z + 1$, where $y = |N(z_{j_1}^\gamma) \cap Y_\gamma|$ and $z = |N(z_{j_1}^\gamma) \cap Z_\gamma|$. On the other side, we have $\mu(y_{j_0}^\gamma) \leq z + y + 2$ (indeed, $|N(y_{j_0}^e \gamma) \cap Z_\gamma| = z + 1$, $|N(y_{j_0}^\gamma) \cap Y_\gamma| \leq y$ and $|N(y_{j_0}^\gamma) \cap X_\gamma| = 1$). Since $\{y_{j_0}^\gamma, z_{j_1}^\gamma\} \in E'$, it holds that $(K' \backslash \{z_{j_1}\}) \cup \{y_{j_0}\}$ is a safe replacement. After all these replacements, given any $\gamma \in E_1$, $tr(Z_\gamma) \neq \emptyset$ implies that $tr(Y_\gamma) = Y_\gamma$.
We now proceed to replacements between gadgets F_γ, $\gamma \in E_1$. If one can find $a, b \in E_1$ such that $tr(Z_a) \neq \emptyset$ and $tr(Y_b) \neq Y_b$, then let j_0 be such that $y_{j_0}^b \notin tr(Y_b)$ and let j_1 be such that $z_{j_1}^a \in tr(Z_a)$. We have $\mu(z_{j_1}^a) \geq T + 1$ and $\mu(y_{j_0}^b) \leq T - 1$. Thus $(K' \backslash \{z_{j_1}\}) \cup \{y_{j_1}\}$ is a safe replacement.
As previously, the replacements ends either when $tr(Y_\gamma) = Y_\gamma$ for all $\gamma \in E_1$ or when $tr(Z_\gamma) = \emptyset$ for all $\gamma \in E_1$. □

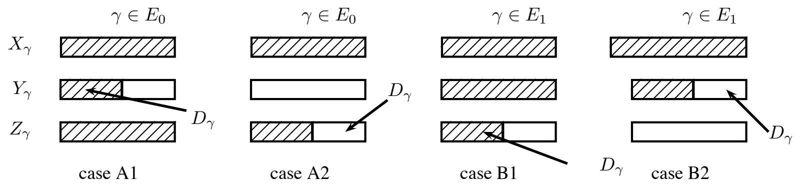

Fig. 2: Schema of different cases. Shaded rectangles represent part of K'.

We now define for each case and each $\gamma \in \mathcal{I}$ the set of vertices $D_\gamma \subseteq Y_\gamma \cup Z_\gamma$ that have to be replaced:

- case A1: for all $\gamma \in E_0$, $D_\gamma = Y_\gamma \cap K'$
- case A2: for all $\gamma \in E_0$, $D_\gamma = Z_\gamma \setminus K'$
- case B1: for all $\gamma \in E_1$, $D_\gamma = Z_\gamma \cap K'$
- case B2: for all $\gamma \in E_1$, $D_\gamma = Y_\gamma \setminus K'$

Notice that if $D_\gamma = \emptyset$ for all $\gamma \in E_0$ (resp. for all $\gamma \in E_1$), then cases A1 and A2 (resp. B1 and B2) collapse. If such a case happen for all $\gamma \in \mathcal{I}$, we can immediately conclude, as we will see in Lemma 8. Now, we will show that if cases A1 and B1 happen (or if $D_\gamma = \emptyset$ for all $\gamma \in \mathcal{I}$), then the solution must hit the clique A in only one subclique A^i for some $i \in \{1, ..., t\}$:

Lemma 7. *If cases A1 and B1 happen (or if $D_\gamma = \emptyset$ for all $\gamma \in \mathcal{I}$), then there exists $i \in \{1, ..., t\}$ such that $tr(A) \subseteq A^i$, i.e. the solution K' only appears in one clique A^i among A.*

Proof. Let $\Delta = \sum_{\gamma \in \mathcal{I}} |D_\gamma|$, and suppose by contradiction that there exists $i, j \in \{1, ..., t\}$ with $i \neq j$ such that $K' \cap A^i \neq \emptyset$ and $K' \cap A^j \neq \emptyset$. First, since we are in case A1 and B1, the number of edges induced by each gadget is at least $\binom{T}{2}$. Then, let S (resp. $\bar{S}$) be the number of pairs of gadgets corresponding to a bit on which the binary representations of i and j is the same (resp. differ). Recall that $S + \bar{S} = Mq$. Then, since $i \neq j$, the binary representations of i and j must differ on at least one bit, which implies $\bar{S} \geq M$. Let us count the number of edges induced by each pair of gadget, whether they correspond to a bit value shared by the binary representation of i and j or not.
Let $p \in \{1, ..., q\}$ such that the binary representations of i and j are the same. Then, for all $h \in \{1, ..., M\}$, three cases may happen:

- $Y_{\alpha_p^h} \subseteq K'$ and $Y_{\beta_p^h} \subseteq K'$. In this case the pair of gadgets induces at least $2\binom{T}{2} + 2$ edges.
- $Y_{\alpha_p^h} \subseteq K'$ and $Z_{\beta_p^h} \subseteq K'$ (or the contrary). In this case the pair of gadgets induces at least $2\binom{T}{2} + 1$ edges.
- $Z_{\alpha_p^h} \subseteq K'$ and $Z_{\beta_p^h} \subseteq K'$. In this case the pair of gadgets induces at least $2\binom{T}{2} + T$ edges, since either $Z_{\alpha_p^h}$ or $Z_{\beta_p^h}$ is adjacent to at least one vertex of $tr(A^i)$.

Hence, in all three cases the solution in each pair of such gadgets induces at least $(2\binom{T}{2} + 1)$ edges.

Let us now focus on some $p \in \{1, ..., q\}$ such that the binary representations of i and j differ. Then, for all $h \in \{1, ..., M\}$, notice that both $Z_{\alpha_p^h}$ and $Z_{\beta_p^h}$ are adjacent to at least one vertex in $A^i \cup A_j$. Here again three cases may happen:

- $Y_{\alpha_p^h} \subseteq K'$ and $Y_{\beta_p^h} \subseteq K'$. In this case the pair of gadgets induces at least $2\binom{T}{2} + 2$ edges.
- $Y_{\alpha_p^h} \subseteq K'$ and $Z_{\beta_p^h} \subseteq K'$ (or the contrary). In this case the pair of gadgets induces at least $2\binom{T}{2} + T + 1$ edges.
- $Z_{\alpha_p^h} \subseteq K'$ and $Z_{\beta_p^h} \subseteq K'$. In this case the pair of gadgets induces at least $2\binom{T}{2} + 2T$ edges.

Hence, in all three cases the solution in each pair of such gadgets induces at least $(2\binom{T}{2} + 2)$ edges. In addition, it is easily seen that the number of edges induced by $tr(A)$ is $\binom{T}{2} - \binom{\Delta}{2} - \Delta(T - \Delta)$, since it is a clique of size $(T - \Delta)$. To resume:

- $tr(A)$ induces $(\binom{T}{2} - \binom{\Delta}{2} - \Delta(T - \Delta))$ edges.
- Each gadget (both from the edge or the instance selector) induces at least $\binom{T}{2}$ edges (there are $(m + 2Mq)$ gadgets), and more precisely:

- Each pair of gadgets corresponding to a shared bit value of the binary representation of i and j induces $(\binom{T}{2}+1)$ edge (*i.e.* one more than the "normal" ones). There are S such pairs of gadgets.
- Each pair of gadgets corresponding to a different bit value of the binary representation of i and j induces $(\binom{T}{2}+2)$ edge (*i.e.* two more than the "normal" ones). There are $\bar{S}$ such pairs of gadgets.

Thus we have:

$$\begin{aligned} E(K') &\geq \binom{T}{2} - \binom{\Delta}{2} - \Delta(T-\Delta) + \binom{T}{2}(m+2Mq) + S + 2\bar{S} \\ &= \binom{T}{2} - \binom{\Delta}{2} - \Delta(T-\Delta) + \binom{T}{2}(m+2Mq) + Mq + \bar{S} \\ &\geq \binom{T}{2} - \binom{\Delta}{2} - \Delta(T-\Delta) + \binom{T}{2}(m+2Mq) + Mq + M \end{aligned}$$

And thus

$$E(K') - C' \geq M - m + \binom{k}{2} - \binom{\Delta}{2} - \Delta(T-\Delta)$$

Since $M = n^6$, we have $E(K') > C'$ which is impossible. □

Lemma 8. *If $D_\gamma = \emptyset$ for all $\gamma \in \mathcal{I}$, then there exists $i \in \{1,...,t\}$ such that G contains a clique of size k.*

Proof. By construction, we have $|tr(A)| = T$ and $|tr(F_\gamma)| = 2T$ for all $\gamma \in \mathcal{I}$. Thus, $E(tr(A)) = \binom{T}{2}$ and $E(tr(F_\gamma)) = \binom{T}{2}+1$ if $\gamma \in E_1$, and $E(tr(F_e)) = \binom{T}{2}$ if $\gamma \in E_0$. Hence, we have $E(K') \geq \binom{T}{2} + \binom{T}{2}(m+2Mq) + |E_1|$.
By Lemma 7, there exists $i \in \{1,...,t\}$ such that $tr(A) \subseteq A^i$. Thus, there are at most Mq gadgets among the instance selector which are not adjacent to $tr(A)$, and which can belong to E_0. This implies that there are at least Mq gadgets among the instance selector which must belong to E_1. Let $E_0^e = E_0 \cap \{1,...,m\}$ be the restriction of E_0 in the edge selector, and similarly $E_1^e = \{1,...,m\} \setminus E_0^e$. The arguments above show that $|E_1^e| \leq m - \binom{k}{2}$, which implies $|E_0^e| \geq \binom{k}{2}$. In addition, each gadget $j \in E_0^e$ corresponding to the edge $e_j = \{u,v\}$ of G_i is adjacent to the cliques A_u^i and A_v^i, which must be such that $A_u^i \cap K' = \emptyset$ and $A_v^i \cap K' = \emptyset$ by definition of E_0. However, since $|tr(A)| = |tr(A^i)| = T$, the number of such cliques is at most $n - \lfloor \frac{T}{n} \rfloor = k$. This proves that these $|E_0^e|$ edges of G can be induced by at most k vertices, *i.e.* G_i contains a clique of size k. □

Let us now combine the four possible cases of Lemmas 5 and 6:

– Case A1 and B1: let $\Delta = \sum_{\gamma \in \mathcal{I}} |D_\gamma|$, and suppose that $\Delta > 0$ (otherwise we conclude by Lemma 8). Let us count the number of edges induced by such a solution. To do so, we count the number of edges induced by the solution among vertices of A, and the number of edges covered by the solution among the gadgets. First, it is clear that $|tr(A)| = T - \Delta$, and thus the number of edges induced by $tr(A)$ is $(\binom{T}{2} - \binom{\Delta}{2} - \Delta(T-\Delta))$ since A is a clique. In addition, since we are in case A1 and B1, the trace of the solution in all gadgets (both from the edge or the instance selector) covers at least $\binom{T}{2}$ edges. More precisely, for each gadget $\gamma \in \mathcal{I}$ three cases may happen:

- if $D_\gamma = \emptyset$, then $tr(F_\gamma)$ covers exactly $\binom{T}{2}$ edges if $\gamma \in E_0$ and exactly $\binom{T}{2}$ edges if $\gamma \in E_1$.
- if $D_\gamma \neq \emptyset$, then:
  * if $\gamma \in E_0$, then since each vertex of Y_γ is connected to all vertices of Z_γ and to one vertex of X_γ, we have that $tr(F_\gamma)$ covers exactly $(\binom{T}{2} + |D_\gamma|(T+1))$ edges (see Figure 2).
  * if $\gamma \in E_1$, then since each vertex of Z_γ is connected to all vertices of Y_γ, and to at least one vertex of $tr(A)$, we have that $tr(F_\gamma)$ covers at least $(\binom{T}{2} + 1 + |D_\gamma|(T+1))$ edges (recall that if D_γ the gadgets covers exactly $(\binom{T}{2} + 1)$ edges).

Summing up to all gadgets, the solution among all gadgets covers $(\binom{T}{2}(m + 2Mq) + |E_1| + \Delta(T+1))$ edges.
We define $E_0^e = E_0 \cap \{1, ..., m\}$ the restriction of E_0 to the edge selector and $E_0^b = E_0 \setminus E_0^e$ the restriction of E_0 to the instance selector, as well as the corresponding sets $E_1^e = E_1 \cap \{1, ..., m\}$ and $E_1^b = E_1 \setminus E_1^e$.
By Lemma 7, there exist $i \in \{1, ..., t\}$ such that $tr(A) \subseteq A^i$. This implies that $|E_0^b| = |E_1^b| = Mq$ (roughly speaking, for each pair of gadgets of the instance selector, only one of the two is connected to A_i and thus to $tr(A)$, depending on the corresponding bit value). Thus, we have $|E_1| = Mq + |E_1^e| = Mq + m - |E_0^e|$.
Combining all these, we obtain:

$$E(K') \geq \binom{T}{2} - \binom{\Delta}{2} - \Delta(T-\Delta) + \binom{T}{2}(m+2Mq) + \Delta(T+1) + Mq + m - |E_0^e|$$

And thus:

$$\begin{aligned} E(K') - C' &\geq \binom{k}{2} + \Delta(T+1) - |E_0^e| - \binom{\Delta}{2} - \Delta(T-\Delta) \\ &= \frac{\Delta(\Delta+3)}{2} + \binom{k}{2} - |E_0^e| \end{aligned}$$

Thus, since we supposed $E(K') - C' \leq 0$ it implies

$$|E_0^e| \geq \frac{\Delta(\Delta+3)}{2} + \binom{k}{2} \tag{1}$$

On the other hand, the number of vertices of G_i inducing all edges of E_0^e is at most $k + \lfloor \frac{\Delta}{n} \rfloor$. Hence we have $|E_0^e| \leq \binom{k+\lfloor \frac{\Delta}{n} \rfloor}{2}$. Hence we have:

$$\binom{k + \lfloor \frac{\Delta}{n} \rfloor}{2} \geq \frac{\Delta(\Delta+3)}{2} + \binom{k}{2}$$

If $\Delta < n$, then $\lfloor \frac{\Delta}{n} \rfloor = 0$ and it contradicts the previous inequality. If $\Delta \geq n$, then it contradicts inequality (1) since we have by definition $|E_0^e| \leq m$. Hence, we must have $\Delta = 0$ and the result follows by Lemma 8.

– Case A2 and B2: let $\Delta_0 = \sum_{\gamma \in E_0} |D_\gamma|$, $\Delta_1 = \sum_{\gamma \in E_1} |D_\gamma|$, and $\Delta = \Delta_0 + \Delta_1$, and suppose that $\Delta > 0$ (otherwise we conclude by Lemma 8). Let us notice that for all $u \in tr(A)$, $\mu(u) \geq T$. On the other hand, for all $\gamma \in \mathcal{I}$ such that there exists $v \in D_\gamma$, we have $\mu(v) \leq T$ (remark that if $\gamma \in E_1$, then $D_\gamma \subseteq Y_\gamma$, and if $\gamma \in E_0$, then v is not adjacent to $tr(A)$ by definition of E_0). Thus $(K' \setminus \{u\}) \cup \{v\}$ is a safe replacement. Since before this replacement we had $tr(A) = T + \Delta$, it is clear that we can repeat this replacement (*i.e.* $K' \setminus \{u\} \cup \{v\}$ where $u \in tr(A)$ and $v \in D_\gamma$ for some $\gamma \in \mathcal{I}$) Δ times safely. At this point, the updated value of Δ is 0, *i.e.* $D_\gamma = \emptyset$ for all $\gamma \in \mathcal{I}$. We then conclude by Lemma 8.

– Case A2 and B1: if there exists $\gamma \in E_0$ such that there exists $u \in D_\gamma$, then $\mu(u) < T$. If such a vertex exists, then either $|tr(A)| > T$ or there exists $\gamma' \in E_1$ such that there exists $v \in D_{\gamma'}$. In the first case for all $x \in tr(A)$ we have $\mu(x) \geq T$, and $(K' \setminus \{x\}) \cup \{u\}$ is a safe replacement. In the second case we have $\mu(v) > T$ and here again $(K' \setminus \{v\}) \cup \{u\}$ is a safe replacement.
  After these replacements we must have $D_\gamma = \emptyset$ for all $\gamma \in E_0$, and we can apply the case A1 and B1.

– Case A1 and B2: if there exists $\gamma \in E_1$ such that there exists $u \in D_\gamma$, then $\mu(u) < T$. If such a vertex exists, then either $|tr(A)| > T$ or there exists $\gamma' \in E_0$ such that there exists $v \in D_{\gamma'}$. In the first case for all $x \in tr(A)$ we have $\mu(x) \geq T$, and $(K' \setminus \{x\}) \cup \{u\}$ is a safe replacement. In the second case we have $\mu(v) > T$ and here again $(K' \setminus \{v\}) \cup \{u\}$ is a safe replacement.
  After these replacements we must have $D_\gamma = \emptyset$ for all $\gamma \in E_1$, and we can apply the case A1 and B1.