The k-Sparsest Subgraph Problem

Rémi Watrigant, Marin Bougeret, Rodolphe Giroudeau

The k-Sparsest Subgraph Problem*

R. Watrigant, M. Bougeret, and R. Giroudeau

LIRMM-CNRS-UMR 5506-161, rue Ada 34090 Montpellier, France

Abstract. Given a simple undirected graph $G = (V, E)$ and an integer $k \leq |V|$, the k-SPARSEST SUBGRAPH problem asks for a set of k vertices that induce the minimum number of edges. As a generalization of the classical INDEPENDENT SET PROBLEM, k-SPARSEST SUBGRAPH cannot admit (unless $\mathcal{P} = \mathcal{NP}$) neither an approximation nor an FPT algorithm (parameterized by the number of edges in the solution) in all graph classes where INDEPENDENT SET is $\mathcal{NP}$-hard . Thus, it appears natural to investigate the approximability and fixed parameterized tractability of k-SPARSEST SUBGRAPH in graph classes where INDEPENDENT SET is polynomial, such as subclasses of perfect graphs. In this paper, we use dynamic programming to design a PTAS in proper interval graph and an FPT algorithm in interval graphs (parameterized by the number of edges in the solution).

1 Introduction and related Problems

1.1 Introduction

Given a graph $G = (V, E)$ and $k \leq |V|$ the k-SPARSEST SUBGRAPH problem (or the K-LIGHTEST SUBGRAPH for the weighted version) asks for a set S of exactly k vertices that minimizes the number of edges of $G[S]$, the subgraph induced by S. As a generalization of the classical INDEPENDENT SET problem (where the number of edges in the induced subgraph is required to be 0), this problem is $\mathcal{NP}$-hard in general graphs. Let us first recall the definition of some related problems, and then discuss their relation to k-SPARSEST SUBGRAPH.

In the MAXIMUM QUASI-INDEPENDENT SET (QIS) problem [5] (also called k-EDGE-IN in [10]), we are given a graph G and an integer C, and we ask for a set of vertices S of maximum size such that $G[S]$ has less than C edges.

In the MINIMUM PARTIAL VERTEX COVER (PVC) problem [11], we are given a graph G and an integer C, and we ask for a set of vertices S of minimum size which covers at least C edges (an edge $\{u, v\}$ is said to be covered by S if either $u \in S$ or $v \in S$).

Finally, we can mention the corresponding maximization problem of k-SPARSEST SUBGRAPH, namely k-DENSEST SUBGRAPH (or k-HEAVIEST SUBGRAPH for the weighted version), that consists in finding a subset S of

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

exactly k vertices that maximizes the number of edges in $G[S]$.

The decision versions of QIS, PVC, and k-SPARSEST SUBGRAPH are polynomially equivalent. Indeed, QIS could be considered as a dual version of k-SPARSEST SUBGRAPH where the budget (the number of edges in the solution of k-SPARSEST SUBGRAPH) is fixed. PVC and k-SPARSEST SUBGRAPH are also polynomially equivalent as for any S, the number of edges in $G[S]$ plus the number of edges covered by $V \backslash S$ equals $|E|$. Finally, it is obvious that any exact result for k-DENSEST SUBGRAPH on a graph class immediately transfers to k-SPARSEST SUBGRAPH for the complementary class (and conversely).

As a consequence, the complexity status of k-SPARSEST SUBGRAPH is already known in several subclasses of perfect graphs, namely in co-comparability and co-chordal graphs for $\mathcal{NP}$-completeness, and in split graphs and in trees for polynomial algorithms. We believe that the $\mathcal{NP}$-hardness in interval graphs may be a tough question, as for example the complexity in bipartite graphs is still currently studied [2], and the complexity of k-DENSEST SUBGRAPH on interval graphs (and even proper interval graphs) is a classical three decades open question raised in [9]. Notice that despite this open question, a PTAS has been designed for k-DENSEST SUBGRAPH in interval graphs in [16].

1.2 Motivation and contributions

Unlike polynomial or $\mathcal{NP}$-hardness results, approximation results on k-DENSEST SUBGRAPH do not directly transfer to k-SPARSEST SUBGRAPH not PVC. Moreover, the approximability status of k-SPARSEST SUBGRAPH did not receive as much attention as the one of k-DENSEST SUBGRAPH. Indeed, k-SPARSEST SUBGRAPH is clearly inapproximable (unless $\mathcal{P} = \mathcal{NP}$) on any class where INDEPENDENT SET is $\mathcal{NP}$-hard, as the optimal value of k-SPARSEST SUBGRAPH is 0 whenever k is lower than the maximum independent set of the input graph. Thus, it appears natural to investigate the approximability of k-SPARSEST SUBGRAPH in graph classes where INDEPENDENT SET is polynomial, such as subclasses of perfect graphs.

In this paper we provide a $PTAS$ in proper interval graphs, and an FPT algorithm in (general) interval graphs parameterized by C, the number of edges in the solution (notice that the last result implies an FPT algorithm for the QIS problem with standard parametrization by k, as well as an FPT algorithm for PVC parameterized by $n-k$).

The intuition of parameterizing by C is that k-SPARSEST SUBGRAPH becomes easy when looking for a solution of cost 0 (as it corresponds to find an independent set). This motivates the design of an efficient algorithm for small C values. Moreover, parameterization by C is stronger than the natural parameterization by k, as we always have $C \leq \binom{k}{2}$.

Graphs classes	k-DENSEST SUBGRAPH	k-SPARSEST SUBGRAPH	PVC
general	$n^{\frac{1}{4}+\epsilon}$-approx [4]	$\mathcal{O}(n^{1-\epsilon})$-inapproximable	2-approx[7] $W[1]-hard$ [11] $\mathcal{O}^*(1,396^C)$ [12]
bip./comp./chordal	$\mathcal{NP}c$ [9]	OPEN	OPEN
co-(bip/comp/chordal)		$\mathcal{NP}c$ (*c.f.* k-densest)	$\mathcal{NP}c$ (*c.f.* k-sparsest)
perfect	$\mathcal{NP}c$ (*c.f.* chordal)	$\mathcal{NP}c$ (*c.f.* k-densest in chordal)	$\mathcal{NP}c$ (*c.f.* k-sparsest)
line	OPEN	$\mathcal{NP}c$ (*c.f.* PVC)	$\mathcal{NP}c$ [1]
cubic	OPEN	$\mathcal{NP}c$ [17]	$\mathcal{NP}c$ (*c.f.* k-sparsest)
trees/cographs/split/ bounded tw/max deg. 2	$\mathcal{P}$ [9]	$\mathcal{P}$ [6]	$\mathcal{P}$ (*c.f.* k-sparsest)
co-(trees/split/bounded tw/max deg. 2)	OPEN	$\mathcal{P}$ (*c.f.* k-densest)	$\mathcal{P}$ (*c.f.* k-sparsest)
clique path	$\mathcal{P}$ [14]	OPEN	OPEN
co-(clique path)		$\mathcal{P}$ (*c.f.* k-densest)	$\mathcal{P}$ (*c.f.* k-sparsest)
σ-quasi elimination order	σ-approx [8]	OPEN	OPEN
chordal	3-approx [15]	OPEN	OPEN
permutation	3/2-approx [3]	OPEN	OPEN
clique star	PTAS [13]	OPEN	OPEN
interval	OPEN, PTAS [16]	OPEN, $FPT(C^*)$ (this paper)	OPEN, $FPT(n-k)$ (*c.f.* k-sparsest)
proper interval	OPEN, PTAS [16] 3/2-approx [3]	OPEN, $PTAS$ (this paper)	OPEN

Fig. 1: Main results for k-DENSEST SUBGRAPH, k-SPARSEST SUBGRAPH and PVC. co-$\mathcal{C}$ denotes the complementary class of $\mathcal{C}$. Clique path (resp. star) denotes the class of graphs whose clique graph is a path (resp. star). σ-quasi elimination order is a generalization of perfect elimination orders for chordal graphs.

2 Preliminaries

Interval graphs are the intersection graph of a set of intervals on the real line. For a set of intervals, the associated intersection graph has one vertex for each interval, and an edge between two vertices corresponding to intervals I_1 and I_2 if and only if I_1 overlaps I_2. A graph is a proper interval graph if it is the intersection graph of a set of intervals on the real line such that no interval properly contains any other interval. As the intersection model of an interval graph can be obtained in polynomial time, we will make no distinction between a vertex and its corresponding interval, as well as we will make no distinction between edges in the graph and overlaps in the corresponding interval model.

For the rest of the paper, $G = (V, E)$ will denote the input graph of the problem, and we define as usually $n = |V|$, $m = |E|$. The associated interval set will be denoted by $\mathcal{I} = \{I_1, ..., I_n\}$. Without loss of polynomiality, we suppose that all endpoints are pairwise distinct. Given $I \in \mathcal{I}$, we denote by $right(I) \in \mathbb{R}$ (resp. $left(I) \in \mathbb{R}$) the right (resp. left) endpoint of I. By extension, for any set $S \subseteq \mathcal{I}$, we define $left(S) = \arg\min_{I \in S} left(I)$ (resp. $right(S) = \arg\max_{I \in S} right(I)$). Unless otherwise stated, we suppose that $\mathcal{I}$ is sorted according to the right endpoints of the intervals (*i.e.* for all $i \in \{2, ..., n\}$ we have $right(I_{i-1}) < right(I_i)$). For $S \subseteq \mathcal{I}$ and $r \leq |S|$, we define the "*r-leftmost intervals of S*" as the r first intervals in an ordering of S (where intervals are sorted according to their right endpoints). Notice that the 1-leftmost interval will simply be called *the leftmost interval*. Given a set $S \subseteq \mathcal{I}$, we denote $cost(S)$ the number of edges in the graph induced by intervals of S.
Finally, we refer the reader to the classical literature for definitions of approximation and FPT algorithms.

3 FPT Algorithm by Dynamic Programming

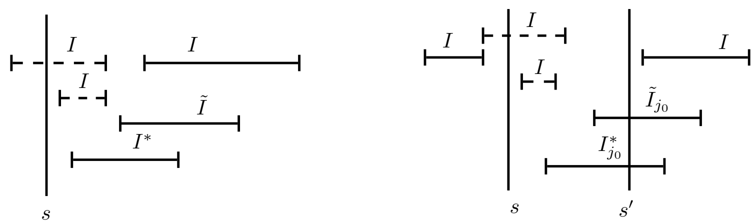
The objective of this section is to provide an FPT algorithm for the k-SPARSEST SUBGRAPH on general (*i.e.* non proper) interval graphs, parameterized by the cost of the solution.

3.1 Preliminaries

Given $x \in \mathbb{R}$ we define $\mathcal{I}_{\geq x} = \{I \in \mathcal{I} : x \leq left(I)\}$ the set of intervals that are after x, $\mathcal{I}_{=x} = \{I \in \mathcal{I} : left(I) < x < right(I)\}$ the set of intervals that cross x, and $\mathcal{I}_{\leq x} = \{I \in \mathcal{I} : right(I) \leq x\}$ the set of intervals that are before x.
Let us start with two lemmas that allow us to restructure optimal solutions by "flushing" intervals to the left.

Lemma 1. *Let $S \subseteq \mathcal{I}$ be a solution, and $s \in \mathbb{R}$ such that $left(S) < s < right(S)$ and $S \cap \mathcal{I}_{=s} = \emptyset$. Let $\tilde{I}$ be the leftmost interval of $S \cap \mathcal{I}_{\geq s}$ and I^* be the leftmost interval of $\mathcal{I}_{\geq s}$. Then we can swap $\tilde{I}$ and I^* to get a solution $S' = (S \backslash \{\tilde{I}\}) \cup \{I^*\}$ such that $cost(S') \leq cost(S)$.*

Proof. Let us suppose that $\tilde{I} \neq I^*$, and let $I \in S$ such that $I \neq \tilde{I}, I^*$. We will show that if I overlaps I^*, then it also overlaps $\tilde{I}$. Thus, suppose that I overlaps I^*. By definition of $\tilde{I}$ and S, we have $right(I^*) < right(\tilde{I}) < right(I)$, and since I overlaps I^*, we have $I \in \mathcal{I}_{=right(I^*)}$ and thus I also overlaps $\tilde{I}$ (see Figure 2a).



(a) (b)

Fig. 2: Different positions of interval I in Lemma 1 (Figure (a)) and Lemma 3 (Figure (b)). Dashed intervals represent forbidden positions.

Lemma 2. *Let $S \subseteq \mathcal{I}$ be a solution, $I_{i_1} \in S$ and $s \in \mathbb{R}$ such that:*
(i) I_{i_1} is the leftmost interval of $S \cap \mathcal{I}_{=s}$
(ii) $\exists \tilde{I} \in S \cap \mathcal{I}_{\geq s}$ such that $\tilde{I}$ overlaps I_{i_1}
Let I^* be the leftmost interval of $\mathcal{I}_{\geq s}$. Then, we can swap I^* and $\tilde{I}$ to get a solution $S' = (S \backslash \{\tilde{I}\}) \cup \{I^*\}$ such that $cost(S') \leq cost(S)$.

Proof. Let us suppose that $\tilde{I} \neq I^*$, otherwise the proof is obvious, and let $I \in S$ such that $I \neq \tilde{I}, I^*$. We will show that if I overlaps I^*, then it also overlaps $\tilde{I}$. Thus, suppose that I overlaps I^*. If $I \in \mathcal{I}_{=s}$, then by definition of I_{i_1}, we must have $right(I_{i_1}) < right(I)$, and since $s < left(\tilde{I}) < right(I_{i_1})$, I must overlap $\tilde{I}$. Otherwise if $I \in \mathcal{I}_{\geq s}$, as in the proof of Lemma 1, by definition of $\tilde{I}$ we have $right(I^*) < right(\tilde{I}) < right(I)$, and since I overlaps I^*, we have $I \in \mathcal{I}_{=right(I^*)}$ and thus I also overlaps $\tilde{I}$.

Lemma 3. *Let $S \subseteq \mathcal{I}$ be a solution and $s, s' \in \mathbb{R}$ with $s < s'$ and such that $\forall I \in S$ we have $right(I) \notin [s, s']$. Let $\tilde{X} = S \cap \mathcal{I}_{\geq s} \cap \mathcal{I}_{=s'}$ and X^* be the $|\tilde{X}|$-leftmost intervals of $\mathcal{I}_{\geq s} \cap \mathcal{I}_{=s'}$. Then we can swap $\tilde{X}$ and X^* to get a solution $S' = (S \backslash \tilde{X}) \cup X^*$ such that $cost(S^*) \leq cost(\tilde{S})$.*

Proof. We suppose that $\tilde{X} \neq X^*$ and that both sets are non empty. Let $\tilde{X} = \{\tilde{I}_1, ..., \tilde{I}_{|\tilde{X}|}\}$, and $X^* = \{I^*_1, ..., I^*_{|\tilde{X}|}\}$. We suppose moreover that for all $j \in \{2, ..., |\tilde{X}|\}$ we have $right(\tilde{I}_{j-1}) < right(\tilde{I}_j)$ and $right(I^*_{j-1}) < right(I^*_j)$ (*i.e.* $\tilde{X}$ and X^* are sorted by their right endpoints). Let j_0 be the minimum index such that $\tilde{I}_{j_0} \neq I^*_{j_0}$, and let $I \in S \backslash (\tilde{X} \cup X^*)$ (we thus have $right(I^*_{j_0}) < right(\tilde{I}_{j_0})$). We will show that if I overlaps $I^*_{j_0}$, then I also overlaps $\tilde{I}_{j_0}$. To do so, suppose that I overlaps $I^*_{j_0}$, and let us distinguish between two cases (see Figure 2b). If $s' < right(I)$, then since $right(I^*_{j_0}) < right(\tilde{I}_{j_0})$, it is clear that I also overlaps $\tilde{I}_{j_0}$. Otherwise, if $right(I) < s'$, then by definition $right(I) < s$, and thus I cannot overlap $I^*_{j_0}$.

3.2 Algorithm

Recall that our objective is to prove that the decision problem "given an instance $(\mathcal{I}, k)$ of k-SPARSEST SUBGRAPH, does $Opt(\mathcal{I}, k) \leq C^*$?", is FPT parametrized by C^*.
We construct in Algorithm 1 a dynamic programming algorithm that given any $next \in \mathbb{R}$, $t \leq k$ and $C \leq C^*$ returns a set S of t vertices in $\mathcal{I}_{\geq next}$ of cost at most C if it was possible, and returns NO otherwise.
We define $\Omega_{next}(C) \subseteq \mathcal{P}(\mathcal{I})$ (where $\mathcal{P}(\mathcal{I})$ is the set of all subsets of $\mathcal{I}$) such that for all $T \in \Omega_{next}(C)$ we have:
- $G[T]$ is connected
- $cost(T) \leq C$
- $left(T) = left(\mathcal{I}_{\geq next})$

Roughly speaking, $\Omega_{next}(C)$ is the set of all connected components of cost at most C that start immediately after $next$. Given $next$ and t, the algorithm branches on a subset of $\Omega_{next}(C)$ (namely $\Gamma_{next}(C)$) to find what could be the next optimal connected component, and then invokes a recursive call.
We prove in Lemma 4 that each $T \in \Omega_{next}(C)$ can be restructured into a "well-structured" component of smaller cost, and in Lemma 5 that the size of the set of all "well-structured" components ($\Gamma_{next}(C)$) can be enumerated in FPT time.

Algorithm 1 $DP(next, t, C)$

// For the sake of clarity we drop the classical operations related to the "marking
// table" that avoid multiple computations with same arguments
build $\Gamma_{next}(C)$ (see Definition 1)
if $\Gamma_{next}(C) = \emptyset$ **then**
  return NO
else if $\exists T \in \Gamma_{next}(C)$ with $|T| \geq t$ **then**
  return t vertices of T
else
  return $\arg\min_{C \in \Gamma_{next}(C)} [cost(T) + cost(DP(right(T), t - |T|, C - cost(T)))]$
end if

Let us now show how to restructure a connected component of a given solution. As one could expect, the idea is to apply the domination rules of Lemmas 1, 2 and 3 that consist in "flushing" the intervals to the left. Thus, for any connected component T, we define (recursively on s) $restruct(s, T, i)$ that turns $T \cap \mathcal{I}_{\geq s}$ (the part of T which is after s) into a well structured solution (see Algorithm 2 and Definition 1). Notice that the parameter i and the y_i values will be used in Lemma 5 to show that the output of the algorithm can be encoded in an efficient way.

Definition 1. *Given s and $T \in \Omega_s(C)$, we define:*
- *$WSS(T) = restruct(start(T), T, 0)$ the Well Structured Solution corresponding to T, where $start(T)$ is defined as the point after the left endpoint of the leftmost interval of T (see Figure 3).*
- *$\Gamma_s(C) = \{WSS(T), T \in \Omega_s(C)\}$ the set of well structured connected component of cost at most C that starts just after s.*

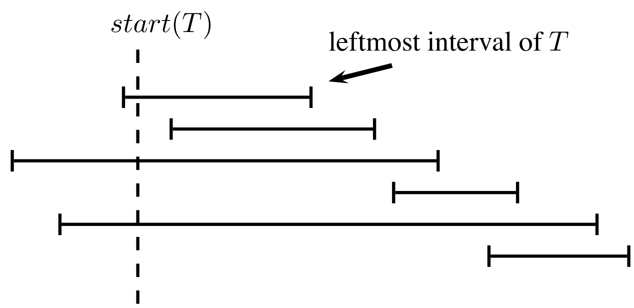


Fig. 3: Example of a connected component T and its corresponding $start(T)$

Remark 1. Notice that at each step of the dynamic programming we branch on $\Gamma_{next}(cost)$, which is the set of all restructured connected component T such that $left(C) = left(\mathcal{I}_{\geq next})$. By Lemma 1, we can suppose that for all optimal solution S^*, we have $left(S^* \cap \mathcal{I}_{\geq next}) = left(\mathcal{I}_{\geq next})$. Roughly speaking, we can suppose that for all optimal solution, the connected component that starts after $\mathcal{I}_{\geq next}$ contains the leftmost interval of $\mathcal{I}_{\geq next}$. As a consequence, the $start(T)$ in Definition 1 forces I_{i_1} to be this leftmost interval.

Lemma 4. *For any s and any $T \in \Omega_s(C)$ we have*
- *$|WSS(T)| = |T|$, i.e. the restructured set has same size*
- *$right(WSS(T)) \leq right(T)$*
- *$cost(WSS(T)) \leq cost(T)$.*

Proof. The first item is clearly true as we only swap sets of intervals of same size. The second item is true as all swapping arguments shift intervals to the left. Let us now turn to the last item. Notice that in the two cases where $restruct$ modifies T, the hypothesis of Lemmas 2 and 3 are verified. Thus, according to these Lemmas the cost of the solution cannot increase.

Lemma 4 confirms that the dynamic programming algorithm can only branch on $\Gamma_s(C)$, avoiding thus branching on $\Omega_s(C)$. It remains now to prove that the dynamic programming algorithm is FPT.

Lemma 5. *For any s, $|\Gamma_s(C)| \leq (\sqrt{2C} + 2)^{C+1}$.*

Algorithm 2 $restruct(s, T, i)$

---

if $T \cap \mathcal{I}_{\geq s} \neq \emptyset$ **then**
  $I_{i_1} \leftarrow$ leftmost interval of $\mathcal{I}_{=s} \cap T$
  // I_{i_1} is always defined, as in the first call s is set to $start(T)$
  if $\nexists I \in T \cap \mathcal{I}_{\geq s}$ which overlaps I_{i_1} **then**
    $y_i \leftarrow 0$
    $restruct(right(I_{i_1}), T, i+1)$
  else
    // we restructure a first interval using Lemma 2
    $\tilde{I} \leftarrow$ leftmost interval of $T \cap \mathcal{I}_{\geq s}$ which overlaps I_{i_1}
    $I^* \leftarrow$ leftmost interval of $\mathcal{I}_{\geq s}$ which overlaps I_{i_1}
    $T \leftarrow (T \backslash \{\tilde{I}\}) \cup \{I^*\}$
    // we restructure a set of intervals using Lemma 3
    $s' \leftarrow \min(right(I^*), right(I_{i_1}))$
    $\tilde{X} \leftarrow T \cap \mathcal{I}_{\geq s} \cap \mathcal{I}_{=s'}$
    $X^* \leftarrow |\tilde{X}|$-leftmost intervals of $\mathcal{I}_{\geq s} \cap \mathcal{I}_{=s'}$
    $T \leftarrow (T \backslash \tilde{X}) \cup X^*$
    $y_i \leftarrow |X^*| + 1$
    $restruct(s', T, i+1)$
  end if
end if

Proof. Let $T \in \Omega_s(C)$, and $WSS(T)$ the associated restructured solution. The key argument is to remark that $WSS(T)$ is entirely determined by the y_i values defined in the *restruct* algorithm. Thus, to each restructured solution $WSS(T)$ we associate the vector $Y(WSS(T)) = (y_0, \ldots, y_{l_{max}})$. Then, the dynamic program will enumerate $\Gamma_s(C)$ by enumerating the set $Y = \{Y(WSS(T)), T \in \Omega_s(C)\}$ of all possible Y vectors.
Notice first that for any i we have $y_i \leq \sqrt{2C} + 2$. Indeed, in the two possible cases of the restructuration ($s' = right(I^*)$ or $s' = right(I_{i_1})$) the $|X^*|$ intervals all overlap s', corresponding to the right endpoint of another interval (I^* or I_{i_1}). Thus, there is at least a clique of size $y_i = |X^*| + 1$ in the solution, whose cost is lower than C.
It remains now to bound l_{max}, the length of the Y vector.
To do that, we show that for any step $i \in \{0, ..., l_{max} - 1\}$ and corresponding s, we can find $I \in \mathcal{I}_{=s}$ and $I' \in \mathcal{I}_{\geq s}$ such that I and I' overlaps, and such that in the next recursive call (with parameter s'), either I or I' belongs to $\mathcal{I}_{\leq s'}$, avoiding multiple counts of same pairs, and implying that $C \geq l_{max} - 1$. Let $i \in \{0, ..., l_{max} - 1\}$. If $y_i \neq 0$, then by definition of I^*, I_{i_1} and I^* are overlapping. Then, since the next recursive call has parameter $s' = \min\{right(I^*), right(I_{i_1})\}$, either I^* or I_{i_1} belongs to $\mathcal{I}_{\leq s'}$. If $y_i = 0$, then $s' = right(I_{i_1})$, and as $i \neq l_{max}$, we know that there exists $I_{i_2} \in \mathcal{I}_{=s'}$ implying that I_{i_2} overlaps I_{i_1}. Finally, it is clear that $I_{i_1} \in \mathcal{I}_{\leq s'}$.

Theorem 1. *k-SPARSEST SUBGRAPH can be solved in* $\mathcal{O}(n^2.k^3.C^*.\,(\sqrt{2C^*} + 2)^{C^*+1})$.

Proof. The dynamic programming algorithm has at most $n.k.C^*$ different inputs. Given fixed parameters, it runs in $\mathcal{O}(|\Gamma_s(C^*)|.k^2 n)$. Indeed, given a Y vector, the corresponding connected component can be built in $\mathcal{O}(l_{max} n) \subseteq \mathcal{O}(C^* n) \subseteq \mathcal{O}(k^2 n)$ as for any $i \leq l_{max}$ it takes $\mathcal{O}(n)$ to find the corresponding y_i intervals.

4 PTAS for Proper Intervals Graphs

In this section we design a PTAS for k-SPARSEST SUBGRAPH in proper interval. We first assume that the instance has one connected component. We prove that we can re-structure an optimal solution Opt into a near optimal solution Opt' such that the pattern used in Opt' in each "block" (a block corresponds to a subset of consecutive intervals in the input) is simple enough to be enumerated in polynomial time. Then, a dynamic programming algorithm will process the graph blocks by blocks from left to right and enumerate for each one all the possible patterns.

4.1 Definitions

Let us define some notation that will be used in the algorithm. Recall that we are now given a set of proper intervals $\mathcal{I} = \{I_1, ..., I_n\}$ sorted by their right endpoints (and by their left endpoints equivalently).

First, we define by induction the following decomposition of the input graph: Let $I_{m_1} = I_1$, $L_1 = I_{m_1}$, $R_1 = \{I_j, j > m_1, I_j \text{ overlaps } I_{m_1}\}$. Then, given any $i \geq 1$ we define (while there remains some intervals after R_i):

- $I_{m_{i+1}}$ is the rightmost interval of the set $X = \{I \notin R_i, \exists I' \in R_i$ s.t. I overlaps $I'\}$ (X is well defined as the instance has a unique connected component)
- $L_{i+1} = \{I_j, j \leq m_{i+1}, I_j \text{ overlaps } I_{m_{i+1}} \text{ and } I_j \notin R_i\}$
- $R_{i+1} = \{I_j, j > m_{i+1}, I_j \text{ overlaps } I_{m_{i+1}}\}$.

Let a denote the maximum i such that I_{m_i} is defined. Notice that R_a may be empty, and that $I_{m_i} \in L_i$ for all $i \in \{1, ..., a\}$.
For any $i \in \{1, ..., a\}$ we define the block i as $B_i = L_i \bigcup R_i$. Thus, the set of intervals is partitioned into blocs B_i for $1 \leq i \leq a$. Such a decomposition is depicted in Figure 4.
For any $1 \leq i \leq a$ and any solution S (a subset of k intervals), let $L_i^S = L_i \bigcap S$, $R_i^S = R_i \bigcap S$, and $B_i^S = B_i \bigcap S$.
Notice that for any S and i, intervals of R_i^S do not intersect intervals of R_{i-1}^S, and intervals of L_i^S do not intersect $I_{m_{i-1}}$ nor intervals of L_{i-1}^S.
We can now write the cost of a solution as the sum of the costs inside the blocks and the costs between the blocks. Thus, we have $cost(S) = \sum_{i=1}^{a} cost(B_i^S) + \sum_{i=1}^{a-1} cost(R_i^S, L_{i+1}^S)$, where $cost(B_i^S)$ is the number of edges in the subgraph induced by B_i^S, and $cost(X_1, X_2) = |\{(I_l, I_{l'}) \in E, I_l \in X_1, I_{l'} \in X_2\}|$. Indeed, by definition, the only edges between blocks B_i and B_{i+1} are edges between R_i and L_{i+1}.

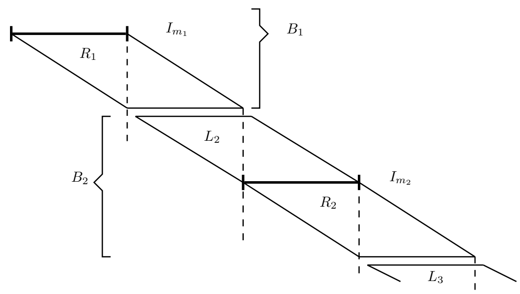


Fig. 4: Schema of the decomposition used in the algorithm.

4.2 Compacting blocks

Let $Comp$ be an injective function from $\mathcal{I}$ to $\mathcal{I}$. For any $S \subseteq \mathcal{I}$, we define $Comp(S) = \bigcup_{I \in S} Comp(I)$. The function $Comp$ is called a *compaction* if for any $S \subseteq \mathcal{I}$ and any $1 \leq i \leq a$ the following holds:

- for all $I \in R_i^S$ we have $Comp(I) \in R_i$ and $right(Comp(I)) \leq right(I)$.
- for all $I \in L_i^S$ we have $Comp(I) \in L_i$ and $right(I) \leq right(Comp(I))$.

Roughly speaking, a compaction "pushes" intervals of B_i^S toward the center I_{m_i}. The idea is that a compaction may increase the cost of a solution inside the blocks, but cannot increase the costs between the blocks. Thus, let us define a ρ-compaction as a compaction $Comp$ such that for any $S \subseteq \mathcal{I}$ and for all $i \in \{1, ..., a\}$ we have $cost(Comp(B_i^S)) \leq \rho.cost(B_i^S)$.

Lemma 6. *If $Comp$ is a ρ-compaction, then for any solution S, $cost(Comp(S)) \leq \rho.cost(S)$.*

Proof. By definition of the decomposition, we have

$$\begin{aligned} cost(Comp(S)) &= \sum_{i=1}^{a} cost(Comp(B_i^S)) + \sum_{i=1}^{a-1} cost(Comp(R_i^S), Comp(L_{i+1}^S)) \\ &\leq \sum_{i=1}^{a} \rho.cost(B_i^S) + \sum_{i=1}^{a-1} cost(Comp(R_i^S), Comp(L_{i+1}^S)) \end{aligned}$$

We now prove that $\sum_{i=1}^{a-1} cost(Comp(R_i^S), Comp(L_{i+1}^S)) \leq \sum_{i=1}^{a-1} cost(R_i^S, L_{i+1}^S)$. Indeed, let $I_R \in R_i^S$ and $I_L \in L_{i+1}^S$ such that I_R and I_L do not overlap. Then by definition of a compaction, we have $right(Comp(I_R)) \leq right(I_R)$ and $left(I_L) \leq left(Comp(I_L))$. Thus, intervals $Comp(I_R)$ and $Comp(I_L)$ do not overlap as well, which proves the result.

According to the previous lemma, we only have now to find compactions that preserve costs inside the blocks. Given a fixed ϵ, the objective is now to define a $(1+\epsilon)$-compaction that has a simple structure.

Lemma 7. *For any fixed $P \in \mathbb{N}$, there is a $(1+\frac{4}{P})$-compaction such that for any X, $Comp(X)$ can be described by $(2P+4)$ variables ranging in $\{0,\ldots,n\}$.*

Proof. According to Lemma 6, we only describe $Comp(X)$ for $X \subseteq B_i$, given any $1 \leq i \leq a$. Let $X = X_L \cup X_R$ with $X_L \subseteq L_i$ and $X_R \subseteq R_i$. We define $x_L = |X_L|$, $x_R = |X_R|$. Moreover, we set $x_L = q_L P + r_L$ (with $r_L < P$) and $x_R = q_R P + r_R$ (with $r_R < P$).
Let us split X_L into P subsets $(G_t^L)_{1\leq t\leq P}$ of consecutive intervals (in the ordering of their right endpoints), with $|G_t^L| = q_L + 1$ for $t \in \{1, ..., r_L\}$ and $|G_t^L| = q_L$ for $t \in \{(r_L+1), ..., P\}$ (see Figure 5). Similarly, we split X_R into P subsets $(G_t^R)_{1\leq t\leq P}$ of consecutive intervals, with $|G_t^R| = q_R+1$ for $t \in \{1, ..., r_R\}$ and $|G_t^R| = q_R$ for $t \in \{(r_R+1), ..., P\}$.
For all $t \in \{1, ..., P\}$, let I_t^L (resp. I_t^R) be the rightmost (resp. leftmost) interval of G_t^L (resp. G_t^R). The principle of the compaction is to flush every intervals of G_t^L (resp. G_t^R) to the right (resp. left). Thus, for $t \in \{1, ..., r_L\}$, $Comp(G_t^L)$ is defined as the (q_L+1)-rightmost intervals I such that $right(I) \leq right(I_t^L)$, and for $t \in \{(r_L+1), ..., P\}$, $Comp(G_t^L)$ is defined as the q_L-rightmost intervals I such that $right(I) \leq right(I_t^L)$. Similarly, for $t \in \{1, ..., r_R\}$, $Comp(G_t^R)$ is defined as the (q_R+1)-leftmost intervals I such that $right(I_t^R) \leq right(I)$, and for $t \in \{(r_R+1), ..., P\}$, $Comp(G_t^R)$ is defined as the q_R-rightmost intervals I such that $right(I_t^R) \leq right(I)$. The construction for a block L_i is depicted in Figure 5. It is clear that the mapping $Comp$ described above is a compaction. Moreover, given x_L, r_L, x_R, r_R and I_t^L (resp. I_t^R) for all $1 \leq t \leq P$, we are clearly able to construct $Comp(X)$ in polynomial time. Thus, it remains to prove that $Comp$ is a $(1+\frac{4}{P})$-compaction.

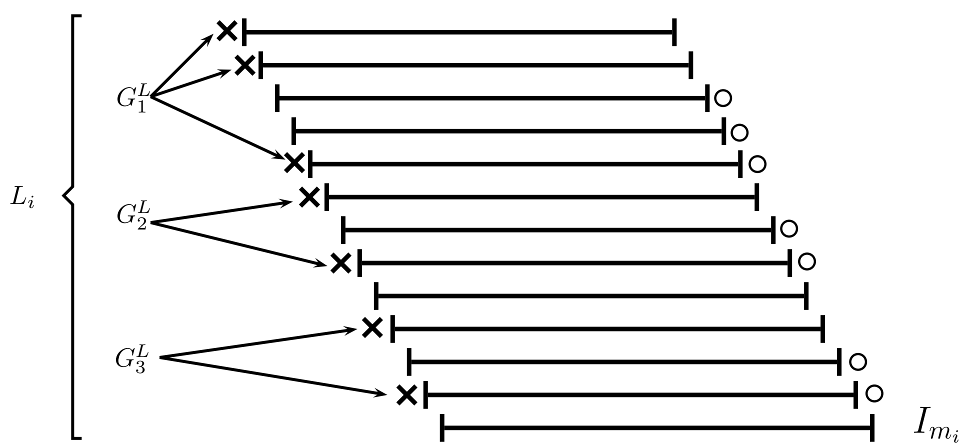


Fig. 5: Exemple of a compaction of a set X for a block L_i, with $P = 3$, and $x_L = 7$. Intervals marked with a cross represent X. Intervals marked with a circle represent $Comp(X)$

The two key arguments are the following:

(i) all intervals of L_i form a clique, as well as all intervals of R_i.

(ii) for any $t_1, t_2 \in \{1, ..., P\}$ with $t_1 \neq P$ and $t_2 \neq 1$, if an interval of $Comp(G^L_{t_1})$ overlaps an interval of $Comp(G^R_{t_2})$, then for any $s_1 \in \{(t_1+1), ..., P\}$ and any $s_2 \in \{1, ..., (t_2-1)\}$, all intervals of $G^L_{s_1}$ overlap all intervals of $G^R_{s_2}$.

For all $t \in \{1, ..., P\}$, we define $x^L_t = |G^L_t| = |Comp(G^L_t)|$, $x^R_t = |G^R_t| = |Comp(G^R_t)|$. By our construction and (i), we have

$$cost(Comp(X)) \leq \binom{x_L}{2} + \binom{x_R}{2} + \sum_{t=1}^{P} cost(Comp(G^L_t), Comp(X) \cap R_i)$$

$$cost(X) \geq \binom{x_L}{2} + \binom{x_R}{2} + \sum_{t=1}^{P} cost(G^L_t, X \cap R_i)$$

Then, for all $t \in \{1, ..., P\}$, let $\lambda_t \in \{0, 1, ..., P\}$ be the maximum s such that an interval of $Comp(G^L_t)$ overlaps an interval of $Comp(G^R_s)$ (we set $\lambda_t = 0$ if no interval of $Comp(G^L_t)$ overlaps an interval of $Comp(G^R_1)$). By (ii), for all $t \in \{1, ..., P\}$, we have $cost(Comp(G^L_t), Comp(X) \cap R_i) \leq x^L_t \sum_{u=1}^{\lambda_t} x^R_u$ and for all $t \in \{2, ..., P\}$, we have $cost(G^L_t, X \cap R_i) \geq x^L_t \sum_{u=1}^{\lambda_{t-1}-1} x^R_u$ (since some intervals of G^L_{t-1} overlap some intervals of $G^R_{\lambda_{t-1}}$, it implies that all intervals of G^L_t overlap all intervals of $G^R_{\lambda_{t-1}-1}$). Combining the previous inequalities, we now have

$$cost(Comp(X)) \leq \binom{x_L}{2} + \binom{x_R}{2} + \sum_{t=1}^{P} x^L_t \sum_{u=1}^{\lambda_t} x^R_u$$

$$cost(X) \geq \binom{x_L}{2} + \binom{x_R}{2} + \sum_{t=2}^{P} x^L_t \sum_{u=1}^{\lambda_{t-1}-1} x^R_u$$

Thus, we have $\Delta = cost(Comp(X)) - cost(X) \leq x^L_1 \sum_{u=1}^{\lambda_1} x^R_u + \sum_{t=2}^{P} x^L_t \sum_{u=\lambda_{t-1}}^{\lambda_t} x^R_u$. As in our case we have $x^L_t \leq (q_L+1)$, we get $\Delta \leq (q_L+1)(\sum_{u=1}^{\lambda_P} x^R_u + \sum_{u=1}^{P} x^R_{\lambda_u}) \leq 2(q_L+1)x_R \leq 2(\frac{x_L}{P}+1)x_R$.

It remains now to handle particular cases, according to the values of x_L and x_R.

- If $x_L \geq P$, then $2(\frac{x_L}{P}+1)x_R \leq \frac{4}{P}x_L x_R$, and $\frac{\Delta}{cost(X)} \leq \frac{\frac{4}{P}x_L x_R}{(x_L-1)x_L+(x_R-1)x_R} \leq \frac{\frac{4}{P}x_L x_R}{\frac{1}{2}(x_L^2+x_R^2)} \leq \frac{4}{P}$ (we lower bounded (x_R-1) by $\frac{x_R}{2}$ as cases with $x_R \leq 1$ lead to even better ratio).
- If $x_L < P$, then we set $Comp(X \cap L_i) = X_L$ (*i.e.* we keep the left part unchanged). If $x_R < P+1$, then we set $Comp(X) = X$ and we get a 1-compaction. Notice that in these cases we are still able to construct $Comp(X)$ in polynomial time. Suppose now that $x_R \geq P+1$. One can improve the previous lower bound and write $cost(X) \geq \frac{(x_L-1)x_L}{2} + \frac{(x_R-1)x_R}{2} + \sum_{t=1}^{P} x^L_t (\sum_{u=1}^{\lambda_t-1} x^R_u)$. Indeed, for all $t \in \{1, ..., x_L\}$ the set G^L_t is a singleton (and $G^L_t = \emptyset$ for $t \in \{x_L+1, ..., P\}$), and thus the interval of G^L_t overlaps some intervals of $G^R_{\lambda_t}$, which implies that it overlaps all intervals of $G^R_{\lambda_t-1}$. Thus, we get $\Delta \leq \sum_{t=1}^{P} x^L_t x^R_{\lambda_t} \leq \sum_{t=1}^{x_L} x^R_{\lambda_t} \leq x_R$, and $\frac{\Delta}{cost(X)} \leq \frac{2x_R}{(x_L-1)x_L+(x_R-1)x_R} \leq \frac{2}{P}$, which terminates the proof of the lemma.

4.3 Algorithm

Let us now write a dynamic programming algorithm for the instances that have a unique connected component (we will drop this hypothesis after). Let Opt be an optimal solution, P a fixed integer and $Comp$ the previous $(1+\frac{4}{P})$-compaction. The algorithm constructs a solution which is at least as good as $Comp(Opt)$ by enumerating for all blocks all the possible compacted patterns (*i.e.* all the possible $Comp(X)$).
Let us now define more formally the algorithm, starting with the parameters. The first parameter $k' \leq k$ is the number of interval to choose. i is the starting block, meaning that the k' interval must be chosen in $\bigcup_{l=i}^{a} B_l$. Finally, B_{i-1}^S represents the set of $2P+4$ variables that encode the set of intervals X_{i-1} chosen in block $(i-1)$. Since we can construct X_{i-1} from B_{i-1}^S in polynomial time, we will directly use B_{i-1}^S to denote X_{i-1}, for the sake of readability.

Algorithm 3 $DP(k', i, B_{i-1}^S)$

// For the sake of clarity we drop the classical operations related to the "marking
// table" that avoid multiple computations with same arguments
// We also drop the base case $i = a+1$ (*i.e.* there are no more remaining intervals in the instance)
$\Omega \leftarrow$ all possible patterns for block i using less or equal than k' intervals
return $\arg\min_{B\in\Omega} cost(B_{i-1}^S \cup B \cup DP(k' - |B|, i+1, B))$

Lemma 8. *For any P, $DP(k, 1, \emptyset)$ outputs a $(1+\frac{4}{P})$-approximation for the k-SPARSEST SUBGRAPH in $\mathcal{O}(n^{\mathcal{O}(P)})$.*

Proof. The objective is to prove that $cost(DP(k, 1, \emptyset)) \leq cost(Comp(Opt))$, where $Comp$ is the previous $(1+\frac{4}{p})$-compaction. According to Lemma 7, it is sufficient to get a $(1+\frac{4}{p})$-approximation.
For sake of readability, for all $i \in \{1, ..., a\}$, we define $B_i^* = Comp(Opt) \cap B_i$ and $k_i^* = |\bigcup_{l=i}^{a} B_i^*|$.
We prove by induction on i (starting from $i = a+1$) that $cost(B_{i-1}^* \cup DP(k_i^*, i, B_{i-1}^*)) \leq cost(Comp(Opt) \cap \bigcup_{l=i-1}^{a} B_l)$.
Let us suppose that the hypothesis is true for $i+1$ and prove it for i. Considering the iteration where DP chooses $B = B_i^*$.

$$cost(B_{i-1}^* \cup DP(k_i^*, i, B_{i-1}^*)) \leq cost(B_{i-1}^*) + cost(B_{i-1}^*, B_i^*) + DP(k_i^* - |B_i^*|, i+1, B_i^*)$$

(recall that $cost(X_1, X_2) = |\{(I_l, I_{l'}) \in E, I_l \in X_1, I_{l'} \in X_2\}|$). Using the induction hypothesis we get the desired result.
The dependency in P in the running time is due to the $n^{2P+\mathcal{O}(1)}$ possible values for the set of parameters and the branching time in $n^{2P+\mathcal{O}(1)}$ when enumerating sets B_i^S.

Finally, let us extend the previous result to instances having several connected component. We only sketch briefly the algorithm as it follows the same idea as, for example, [8] for the k densest.

Let us suppose that for any $k' \leq k$ we have an algorithm $A(k', X)$ which is a ρ-approximation for k'-SPARSEST SUBGRAPH on a instance X having one connected component.
Let $(C_i)_{1 \leq i \leq x}$ denote the connected component of a (general) instance of k-SPARSEST SUBGRAPH. It is sufficient to define a dynamic programming algorithm $DP(k', i)$ that computes a ρ approximation of the k'-SPARSEST SUBGRAPH on $\bigcup_{t=i}^{x}(C_t)$ by keeping the best of all the $\mathcal{A}(l, C_i) + DP(k' - l, i+1)$, for $1 \leq l \leq k'$.
Thus, we get the following result:

Theorem 2. *There is a $PTAS$ for k-SPARSEST SUBGRAPH on proper interval graphs running in $n^{\mathcal{O}(\frac{1}{\epsilon})}$*

5 Conclusion and Future Work

In this paper, we studied the fixed-parameter tractability and approximation of the k-SPARSEST SUBGRAPH problem in subclasses of chordal graphs. More precisely, we designed a $PTAS$ in proper interval graphs and an FPT in interval graphs when parameterized by the cost of the solution. Given that obtaining a negative result for our problem in interval graphs seems to be a tough question, it would be interesting to determine the complexity of the problem in chordal graphs, and then to extend our approximation and fixed parameterized algorithms in case of $\mathcal{NP}$-hardness.

References

1. N. Apollonio and A. Sebő. Minconvex factors of prescribed size in graphs. SIAM Journal of Discrete Mathematics, 23(3):1297–1310, 2009.
2. Nicola Apollonio. Private communication, 2012.
3. J. Backer and J.M. Keil. Constant factor approximation algorithms for the densest k-subgraph problem on proper interval graphs and bipartite permutation graphs. Information Processing Letters, 110(16):635–638, 2010.
4. A. Bhaskara, M. Charikar, E. Chlamtac, U. Feige, and A. Vijayaraghavan. Detecting high log-densities: an $\mathcal{O}(n^{1/4})$ approximation for densest k-subgraph. In Proceedings of the 42nd ACM symposium on Theory of Computing, pages 201–210. ACM, 2010.
5. N. Bourgeois, A. Giannakos, G. Lucarelli, I. Milis, V. Th. Paschos, and O. Pottié. The max quasi-independent set problem. Journal of Combinatorial Optimization, 23(1):94–117, 2012.
6. 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.

7. N. Bshouty and L. Burroughs. Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem. In Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science, pages 298–308. Springer, 1998.
8. 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.
9. D.G. Corneil and Y. Perl. Clustering and domination in perfect graphs. Discrete Applied Mathematics, 9(1):27 – 39, 1984.
10. O. Goldschmidt and D. S. Hochbaum. k-edge subgraph problems. Discrete Applied Mathematics, 74(2):159–169, 1997.
11. J. Guo, R. Niedermeier, and S. Wernicke. Parameterized complexity of vertex cover variants. Theory of Computing Systems, 41(3):501–520, 2007.
12. J. Kneis, A. Langer, and P. Rossmanith. In Proceedings of the 34th Workshop of Graph Theoretic Concepts in Computer Science, pages 240–251. Springer, 2008.
13. M. Liazi, I. Milis, F. Pascual, and V. Zissimopoulos. The densest k-subgraph problem on clique graphs. Journal of Combinatorial Optimization, 14(4):465–474, 2007.
14. M. Liazi, I. Milis, and V. Zissimopoulos. Polynomial variants of the densest/heaviest k-subgraph problem. In Proceedings of the 20th British Combinatorial Conference, Durham, 2005.
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. M. Yannakakis. Node-and edge-deletion NP-complete problems. In Proceedings of the 10th annual ACM Symposium On Theory of Computing, pages 253–264. ACM, 1978.