Polynomial Fixed-parameter Algorithms: A Case Study for Longest Path on Interval Graphs*

Archontia C. Giannopoulou†[1], George B. Mertzios[2], and Rolf Niedermeier[3]

1 **Institute of Informatics, University of Warsaw, Poland**
archontia.giannopoulou@gmail.com
2 **School of Engineering and Computing Sciences, Durham University, UK**
george.mertzios@durham.ac.uk
3 **Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany**
rolf.niedermeier@tu-berlin.de

Abstract

We study the design of fixed-parameter algorithms for problems already known to be solvable in polynomial time. The main motivation is to get more efficient algorithms for problems with unattractive polynomial running times. Here, we focus on a fundamental graph problem: Longest Path; it is NP-hard in general but known to be solvable in $O(n^4)$ time on n-vertex interval graphs. We show how to solve Longest Path on Interval Graphs, parameterized by vertex deletion number k to proper interval graphs, in $O(k^9 n)$ time. Notably, Longest Path is trivially solvable in linear time on proper interval graphs, and the parameter value k can be approximated up to a factor of 4 in linear time. From a more general perspective, we believe that using parameterized complexity analysis for polynomial-time solvable problems offers a very fertile ground for future studies for all sorts of algorithmic problems. It may enable a refined understanding of efficiency aspects for polynomial-time solvable problems, similarly to what classical parameterized complexity analysis does for NP-hard problems.

1998 ACM Subject Classification G.2.2 Graph Theory – Graph Algorithms, Path and Circuit Problems, F.2.2 Nonnumerical Algorithms and Problems – Computations on Discrete Structures

Keywords and phrases fixed-parameter algorithm, preprocessing, data reduction, polynomial-time algorithm, longest path problem, interval graphs, proper interval vertex deletion set

Digital Object Identifier 10.4230/LIPIcs.IPEC.2015.102

1 Introduction

Parameterized complexity analysis [16, 18, 30] is a flourishing field dealing with the exact solvability of NP-hard problems. The key idea is to lift classical complexity analysis, rooted in the P versus NP phenomenon, from a one-dimensional to a two- (or even multi-)dimensional perspective, the key concept being "fixed-parameter tractability (FPT)". But why should this natural and successful approach be limited to intractable (i.e., NP-hard) problems? We are convinced that appropriately parameterizing polynomially solvable problems sheds new light on what makes a problem far from being solvable in *linear* time, in the same way as

* Partially supported by the EPSRC Grant EP/K022660/1 and the Warsaw Center of Mathematics and Computer Science.
† The main part of this paper was prepared while the author was affiliated at the School of Engineering and Computing Sciences, Durham University, UK.

classical FPT algorithms help in illuminating what makes an NP-hard problem far from being solvable in *polynomial* time. In a nutshell, the credo and leitmotif of this paper is that "FPT inside P" is a very interesting, but still too little explored, line of research.

The known results fitting under this leitmotif are somewhat scattered around in the literature and do not systematically refer to or exploit the toolbox of parameterized algorithm design. This should change and "FPT inside P" should be placed on a much wider footing, using parameterized algorithm design techniques such as data reduction and kernelization. As a simple illustrative example, consider the Maximum Matching problem. By following a "Buss-like" kernelization (as is standard knowledge in parameterized algorithmics [16, 30]) and then applying a known polynomial-time matching algorithm, it is not difficult to derive an efficient algorithm that, given a graph G with n vertices, computes a matching of size at least k in $O(kn + k^3)$ time.

More formally, and somewhat more generally, we propose the following scenario. Given a problem with instance size n for which there exists an $O(n^c)$-time algorithm, our aim is to identify appropriate parameters k and to derive algorithms with time complexity $f(k) \cdot n^{c'}$ such that $c' < c$, where $f(k)$ depends only on k. First we refine the class FPT by defining, for every polynomially-bounded function $p(n)$, the class FPT($p(n)$) containing the problems solvable in $f(k) \cdot p(n)$ time, where $f(k)$ is an arbitrary (possibly exponential) function of k. It is important to note that, in strong contrast to FPT algorithms for NP-hard problems, here the function $f(k)$ may also become *polynomial* in k. Motivated by this, we refine the class P by introducing, for every polynomial $p(n)$, the class *P-FPT*($p(n)$) *(Polynomial Fixed-Parameter Tractable)*, containing the problems solvable in $O(k^t \cdot p(n))$ time for some constant $t \geq 1$, i.e., the dependency of the complexity on the parameter k is at most polynomial. In this paper we focus our attention on the (practically perhaps most attractive) subclass *PL-FPT (Polynomial-Linear Fixed-Parameter Tractable)*, where PL-FPT = P-FPT(n). For example, the algorithm we sketched above for Maximum Matching, parameterized by solution size k, belongs to PL-FPT.

In an attempt to systematically follow the leitmotif "FPT inside P", we put forward three desirable algorithmic properties:

1. The running time should have a polynomial dependency on the parameter.
2. The running time should be as close to linear as possible if the parameter value is constant, improving upon an existing "high-degree" polynomial-time (unparameterized) algorithm.
3. The parameter value, or a good approximation thereof, should be computable efficiently (preferably in linear time) for arbitrary parameter values.

In addition, as this research direction is still only little explored, we suggest and follow a focus first on problems for which the best known upper bounds of the time complexity are polynomials of high degree, e.g., $O(n^4)$ or higher.

Related work

Here we discuss previous work on graph problems that fits under the leitmotif "FPT inside P"; however there exists further related work also in other topics such as string matching [6] or Linear Program solving [28].

The complexity of some known polynomial-time algorithms can be easily "tuned" with respect to specific parameters, thus immediately reducing the complexity whenever these parameters are bounded. For instance, in n-vertex and m-edge graphs with nonnegative edge weights, Dijkstra's $O(m+n \log n)$-time algorithm for computing shortest paths can be adapted to an $O(m + n \log k)$-time algorithm, where k is the number of distinct edge weights [27] (also refer to [31]). In addition, motivated by the quest for explaining the efficiency of several

shortest path heuristics for road networks (where Dijkstra's algorithm is too slow for routing applications), the "highway dimension" was introduced [4] as a parameterization helping to do rigorous proofs about the quality of the heuristics. Altogether, the work on shortest path computations shows that even for quasi-linear-time algorithms adopting a parameterized view may be of significant practical interest.

Maximum flow computations constitute another important application area for "FPT inside P". An $O(k^3 n \log n)$-time maximum flow algorithm was presented [22] for graphs that can be made planar by deleting k "crossing edges"; notably, here it is assumed that the embedding and the k crossing edges are given along with the input. An $O(g^8 n \log^2 n \log^2 C)$-time maximum flow algorithm was developed [12], where g is the genus of the graph and C is the sum of all edge capacities; here it is also assumed that the embedding and the parameter g are given in the input. Finally, we remark that multiterminal flow [21] and Wiener index computations [10] have exploited the treewidth parameter, assuming that the corresponding tree decomposition of the graph is given. However, in both publications [21, 10] the dependency on the parameter k is *exponential*.

Our contribution

In this paper, for illustrating the potential algorithmic challenges posed by the "FPT inside P" framework (which seem to go clearly beyond the known "FPT inside P" examples), we focus on Longest Path on Interval Graphs, which is known to be solvable in $O(n^4)$ time [23], and we derive an PL-FPT-algorithm (with the appropriate parameterization) that satisfies all three desirable algorithmic properties described above.

On general graphs, the decision variant of Longest Path is NP-complete and many FPT algorithms have been designed for it, e.g., [5, 13, 26, 35], contributing to the parameterized algorithm design toolkit techniques such as color-coding [5] (and further randomized techniques [13, 26]) as well as algebraic approaches [35]. Longest Path is known to be solvable in polynomial time only on very few non-trivial graph classes [23, 29] (see also [33] for much smaller graph classes). In particular, a few years ago it was shown that Longest Path on Interval Graphs can be solved in polynomial time, providing an algorithm that runs in $O(n^4)$ time [23]. Notably, a longest path in a *proper* interval graph can be computed by a *trivial* linear-time algorithm since every connected proper interval graph has a Hamiltonian path [7]. Consequently, as these two graph classes seem to behave quite differently, it is natural to parameterize Longest Path on Interval Graphs by the size k of a *minimum proper interval (vertex) deletion set*, i.e., by the minimum number of vertices that need to be deleted to obtain a proper interval graph. That is, this parameterization exploits what is also known as "distance from triviality" [20, 17] in the sense that the parameter k measures how far a given input instance is from a trivially solvable special case. As it turns out, one can compute a 4-approximation of k in $O(n + m)$ time for an interval graph with n vertices and m edges. Based on this, we provide a polynomial fixed-parameter algorithm that runs in $O(k^9 n)$ time, thus proving that Longest Path on Interval Graphs is in the class PL-FPT when parameterized by the size of a minimum proper interval deletion set.

To develop our algorithm, we first introduce in Section 2 two data reduction rules on interval graphs. Each of these reductions shrinks the size of specific vertex subsets, called *reducible* and *weakly reducible* sets, respectively. Then, given any proper interval deletion set D of an interval graph G, in Section 3 we appropriately decompose the graph $G \setminus D$ into two collections $\mathcal{S}_1$ and $\mathcal{S}_2$ of reducible and weakly reducible sets, respectively, on which we apply the reduction rules of Section 2. The resulting interval graph $\widehat{G}$ is *weighted* (with weights on its vertices) and has some special properties; we call $\widehat{G}$ a *special weighted interval*

graph with *parameter* κ, where in this case $\kappa = O(k^3)$. Notably, although $\widehat{G}$ has reduced size, it still has $O(n)$ vertices. Then, in Section 4 we present a fixed-parameter algorithm (with parameter κ) computing in $O(\kappa^3 n)$ time the maximum weight of a path in a special weighted interval graph. We note here that such a maximum-weight path in a special weighted interval graph can be directly mapped back to a longest path in the original interval graph. Thus, our algorithm computes a longest path in the initial interval graph G in $O(\kappa^3 n) = O(k^9 n)$ time. In the concluding section, we give a brief outlook.

Notation

We consider finite, simple, and undirected graphs. Given a graph G, we denote by $V(G)$ and $E(G)$ the sets of its vertices and edges, respectively. A graph G is *weighted* if it is given along with a weight function $w : V(G) \to \mathbb{N}$ on its vertices. An edge between two vertices u and v of a graph $G = (V, E)$ is denoted by uv, and in this case u and v are said to be *adjacent*. The *neighborhood* of a vertex $u \in V$ is the set $N(u) = \{v \in V : uv \in E\}$ of its adjacent vertices. Furthermore we denote by $G[S]$ the subgraph of G induced by the vertex set S and we define $G \setminus S = G[V \setminus S]$. A set $S \subseteq V$ induces an *independent set* (resp. a *clique*) in G if $uv \notin E$ (resp. if $uv \in E$) for every pair of vertices $u, v \in S$.

A graph $G = (V, E)$ is an *interval* graph if each vertex $v \in V$ can be bijectively assigned to a closed interval I_v on the real line, such that $uv \in E$ if and only if $I_u \cap I_v \neq \emptyset$, and then the collection of intervals $\mathcal{I} = \{I_v : v \in V\}$ is an *interval representation* of G. The graph G is a *proper interval graph* if it admits an interval representation $\mathcal{I}$ such that $I_u \not\subseteq I_v$ for every $u, v \in V$, and then $\mathcal{I}$ is a *proper interval representation*. Given an interval graph G, a subset $D \subseteq V(G)$ is a *proper interval deletion set* of G if $G \setminus D$ is a proper interval graph. The *proper interval deletion number* of G is the size of the smallest proper interval deletion set. Finally, for any positive integer t, we denote $[t] = \{1, 2, \ldots, t\}$.

2 Data reductions on interval graphs

In this section we present two data reductions on interval graphs. The first reduction (cf. Reduction Rule 1) shrinks the size of a collection of vertex subsets of a certain kind, called *reducible* sets, and it produces a *weighted* interval graph. The second reduction (cf. Reduction Rule 2) is applied to an arbitrary weighted interval graph; it shrinks the size of a collection of another kind of vertex subsets, called *weakly reducible* sets, and it produces a smaller weighted interval graph. Both reductions retain as an invariant the maximum path weight.

In the remainder of the paper we assume that we are given an interval graph G with n vertices and m edges as input, together with an interval representation $\mathcal{I}$ of G, where the endpoints of the intervals are given sorted increasingly. Without loss of generality we assume that the endpoints of all intervals are distinct. For every vertex $v \in V(G)$ we denote by $I_v = [l_v, r_v]$ the interval of $\mathcal{I}$ that corresponds to v, i.e., l_v and r_v are the left and the right endpoint of I_v, respectively. In particular, G is assumed to be given along with the *right-endpoint ordering* σ of its vertices $V(G)$, i.e., $u <_\sigma v$ if and only if $r_u < r_v$ in the interval representation $\mathcal{I}$. Given a set $S \subseteq V(G)$ we denote by $\mathcal{I}[S]$ the interval representation induced from $\mathcal{I}$ on the intervals of the vertices of S. Finally we denote by $\mathbf{span}(S)$ the interval $[\min\{l_v : v \in S\}, \max\{r_v : v \in S\}]$.

It is well-known that an interval graph G is a proper interval graph if and only if G is $K_{1,3}$-free, i.e., if G does not include the claw $K_{1,3}$ with four vertices (cf. Figure 1) as an induced subgraph [32]. It is worth noting here that it is unknown whether a minimum proper interval deletion set of an interval graph G can be computed in polynomial time. However,

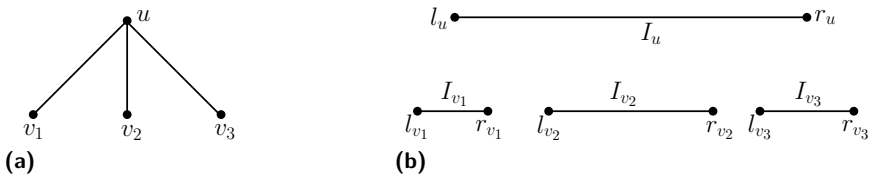

Figure 1 (a) The forbidden induced subgraph (claw $K_{1,3}$) for an interval graph to be a proper interval graph and (b) an interval representation of the $K_{1,3}$.

since there is a unique forbidden induced subgraph $K_{1,3}$ on four vertices, we can apply Cai's generic algorithm [11] on an arbitrary given interval graph G with n vertices to compute a proper interval deletion set of G with minimum size k in FPT time $O(4^k \cdot \text{poly}(n))$. As we prove in the next theorem, a 4-approximation of the minimum proper interval deletion number of an interval graph can be computed much more efficiently.

▶ **Theorem 1.** *Let $G = (V, E)$ be an interval graph, where $|V| = n$ and $|E| = m$. Let k be the size of the minimum proper interval deletion set of G. Then a proper interval deletion set of size at most $4k$ can be computed in $O(n + m)$ time.*

Note that, whenever four vertices induce a claw $K_{1,3}$ in an interval graph G, then in the interval representation $\mathcal{I}$ of G at least one of these intervals is necessarily properly included in another one (e.g., $I_{v_2} \subseteq I_u$ in Figure 1b). However the converse is not always true, as there may exist two vertices u, v in G such that $I_v \subseteq I_u$, although u and v do not participate in any claw $K_{1,3}$ in G. An interval representation $\mathcal{I}$ is called *semi-proper* if, whenever $I_v \subseteq I_u$ in $\mathcal{I}$, then the vertices u and v participate in a claw $K_{1,3}$ in G. Every interval representation $\mathcal{I}$ of a graph G can be efficiently transformed into a semi-proper representation $\mathcal{I}'$ of G, as we prove in the next theorem. In the remainder of the paper we always assume that this preprocessing step has been already applied to $\mathcal{I}$.

▶ **Theorem 2** (preprocessing). *Given an interval representation $\mathcal{I}$, a semi-proper interval representation $\mathcal{I}'$ can be computed in $O(n + m)$ time.*

All our results on interval graphs rely on the notion of a *normal* path [23] (also referred to as a *straight* path in [15, 25]). This notion has also been extended to the greater class of cocomparability graphs [29]. Normal paths are useful in the analysis of our data reductions in this section, as well as in our algorithm in Section 4, as they impose certain *monotonicity* properties of the paths. Informally, the vertices in a normal path appear in a "left-to-right fashion" in the right-endpoint ordering σ. It is known that, for every path P of an interval graph G, there exists a normal path P' on the same vertices as P [23].

▶ **Definition 3.** Let $G = (V, E)$ be an interval graph and σ be a right-endpoint ordering of V. The path $P = (v_1, v_2, \ldots, v_k)$ of G is *normal* if v_1 is the leftmost vertex of $V(P)$ in σ, and if v_i is the leftmost vertex of $N(v_{i-1}) \cap \{v_i, v_{i+1}, \ldots, v_k\}$ in σ, for every $i = 2, \ldots, k$.

The first data reduction

Before we present our Reduction Rule 1, first we introduce the notion of a *reducible* set of vertices.

▶ **Definition 4.** Let G be a (weighted) interval graph and $\mathcal{I}$ be an interval representation of G. A set $S \subseteq V(G)$ is *reducible* if it satisfies the following:

1. $\mathcal{I}[S]$ induces a connected proper interval representation of $G[S]$ and
2. for every $v \in V(G)$ such that $I_v \subseteq \mathbf{span}(S)$ it holds $v \in S$.

The intuition behind reducible sets is as follows. For every reducible set S, a longest path P contains either all vertices of S or none of them. Furthermore, in a longest path P which is *normal* and contains the whole set S, the vertices of S appear *consecutively* in P. Thus we can reduce the number of vertices in a longest normal path P (without changing its total weight) by replacing all vertices of S with a single vertex having weight $|S|$. Now we reduce the interval graph G to the *weighted* interval graph $G^{\#}$ with fewer vertices.

▶ **Reduction Rule 1** (first data reduction). *Let $G = (V, E)$ be an interval graph, $\mathcal{I}$ be an interval representation of G, and D be a proper interval deletion set of G. Let $\mathcal{S}$ be a set of vertex disjoint reducible sets of G, where $S \cap D = \emptyset$, for every $S \in \mathcal{S}$. The* weighted interval graph $G^{\#} = (V^{\#}, E^{\#})$ *is induced by the* weighted interval representation $\mathcal{I}^{\#}$*, which is derived from $\mathcal{I}$ as follows:*

- *for every $S \in \mathcal{S}$, replace in $\mathcal{I}$ the intervals $\{I_v : v \in S\}$ with the single interval $I_S = \mathbf{span}(S)$ which has weight $|S|$; all other intervals receive weight* 1.

In the next theorem we state the correctness of Reduction Rule 1.

▶ **Theorem 5.** *Let ℓ be a positive integer. Then the longest path in G has ℓ vertices if and only if the maximum weight of a path in $G^{\#}$ is ℓ.*

The second data reduction

Before we present our Reduction Rule 2, we introduce the notion of a *weakly reducible* set.

▶ **Definition 6.** Let G be a (weighted) interval graph and $\mathcal{I}$ be an interval representation of G. A set $S \subseteq V(G)$ is *weakly reducible* if it satisfies the following:

1. $\mathcal{I}[S]$ induces a connected proper interval representation of $G[S]$ and
2. for every $v \in V(G)$ and every $u \in S$, if $I_v \subseteq I_u$, then $S \subseteq N(v)$.

The intuition behind weakly reducible sets is as follows. For every weakly reducible set S, a longest path P contains either all vertices of S or none of them. Furthermore, in a longest path P which is *normal* and contains the whole set S, the appearance of the vertices of S in P is *interrupted at most* $|D| + 3$ times by vertices outside S. That is, such a path P has at most $\min\{|S|, |D| + 4\}$ vertex-maximal subpaths with vertices from S. Thus we can reduce the number of vertices in a maximum-weight normal path P (without changing its total weight) by replacing all vertices of S with $\min\{|S|, |D| + 4\}$ vertices; each of these new vertices has the same weight and their total weight sums up to $|S|$. Now we reduce the weighted interval graph G to the weighted interval graph $\widehat{G}$ with fewer vertices.

▶ **Reduction Rule 2** (second data reduction). *Let G be a weighted interval graph with weight function $w : V(G) \to \mathbb{N}$ and $\mathcal{I}$ be an interval representation of G. Let D be a proper interval deletion set of G. Finally, let $\mathcal{S} = \{S_1, S_2, \ldots, S_{|\mathcal{S}|}\}$ be a family of pairwise disjoint weakly reducible sets, where $S_i \cap D = \emptyset$ for every $i \in [|\mathcal{S}|]$. We recursively define the graphs $G_0, G_1, \ldots, G_{|\mathcal{S}|}$ with the interval representations $\mathcal{I}_0, \mathcal{I}_1, \ldots, \mathcal{I}_{|\mathcal{S}|}$ as follows:*

- *$G_0 = G$ and $\mathcal{I}_0 = \mathcal{I}$,*
- *for $1 \leq i \leq |\mathcal{S}|$, $\mathcal{I}_i$ is obtained by replacing in $\mathcal{I}_{i-1}$ the intervals $\{I_v : v \in S_i\}$ with $\min\{|S_i|, |D|+4\}$ copies of the interval $I_{S_i} = \mathbf{span}(S_i)$, each of them having equal weight $\frac{1}{\min\{|S_i|,|D|+4\}} \sum_{u \in S_i} w(u)$; all other intervals remain unchanged, and*
- *finally $\widehat{G} = G_{|\mathcal{S}|}$ and $\widehat{\mathcal{I}} = \mathcal{I}_{|\mathcal{S}|}$.*

Note that in the construction of the interval representation $\mathcal{I}_i$ by Reduction Rule 2, where $i \in [|\mathcal{S}|]$, we can always slightly perturb the endpoints of the $\min\{|S_i|, |D|+4\}$ copies of the interval $I_{S_i} = \mathbf{span}(S_i)$ such that all endpoints remain distinct in $\mathcal{I}_i$, and such that these $\min\{|S_i|, |D|+4\}$ newly introduced intervals induce a proper interval representation in $\mathcal{I}_i$. We are now ready to prove the correctness of Reduction Rule 2.

▶ **Theorem 7.** *Let ℓ be a positive integer. Then the maximum weight of a path in G is ℓ if and only if the maximum weight of a path in $\widehat{G}$ is ℓ.*

3 Special weighted interval graphs

In this section we sequentially apply the two data reductions of Section 2 to a given interval graph G with a proper interval deletion set D. To do so, first we appropriately define a specific family $\mathcal{S}_1$ of *reducible* sets in $G \setminus D$ and we apply Reduction Rule 1 to G with respect to the family $\mathcal{S}_1$, resulting in the weighted interval graph $G^{\#}$. After this operation, D remains a proper interval deletion set of the graph $G^{\#}$. Then we appropriately define a family $\mathcal{S}_2$ of *weakly reducible* sets in $G^{\#} \setminus D$ and we apply Reduction Rule 2 to $G^{\#}$ with respect to the family $\mathcal{S}_2$, resulting to the weighted interval graph $\widehat{G}$. As it turns out, the vertex sets of the union $\mathcal{S}_1 \cup \mathcal{S}_2$ of these two families are a partition of the initial graph $G \setminus D$. Furthermore, Theorems 5 and 7 imply that the longest path in the initial graph G has ℓ vertices if and only if the maximum weight of a path in the final weighted graph $\widehat{G}$ is ℓ. The graph $\widehat{G}$ is then given as input to our parameterized algorithm of Section 4. Now we introduce the crucial notion of a *special weighted interval graph* with *parameter* κ.

▶ **Definition 8** (special weighted interval graph with parameter κ)**.** Let $G = (V, E)$ be a weighted interval graph, $\mathcal{I}$ be an interval representation of G, and $\kappa \in \mathbb{N}$, where V can be partitioned into two sets A and B such that:

1. A is an independent set in G,
2. for every $v \in A$ and every $u \in V \setminus \{v\}$, we have $I_u \not\subseteq I_v$ in $\mathcal{I}$, and
3. $|B| \leq \kappa$.

Then G (resp. $\mathcal{I}$) is a *special weighted interval graph* (resp. *representation*) with *parameter* κ. The partition $V = A \cup B$ is a *special vertex partition* of G.

As we prove in the next theorem, the graph $\widehat{G}$ is a special weighted interval graph and it can be computed efficiently.

▶ **Theorem 9.** *Let $G = (V, E)$ be an interval graph, where $|V| = n$. Let D be a proper interval deletion set of G, where $|D| = k$. Then the graph $\widehat{G} = (\widehat{V}, \widehat{E})$ is a* special weighted interval graph *with* parameter $\kappa = O(k^3)$*. Furthermore, $\widehat{G}$ and a special vertex partition $\widehat{V} = A \cup B$ of $\widehat{G}$ can be computed in $O(k^2 n)$ time.*

Note that, although $\widehat{G}$ is a special weighted interval graph with a parameter κ that depends only on the size of D, it may still have $O(n)$ vertices, as the independent set A in the special vertex partition may be arbitrarily large.

4 Parameterized Longest Path on Interval Graphs

In this section, we first present Algorithm 1 which computes in $O(\kappa^3 n)$ time the maximum weight of a path in a *special weighted* interval graph with parameter κ. Then, using Algorithm 1 and the results of Sections 2 and 3, we conclude with our fixed-parameter algorithm for Longest Path on Interval Graphs, where the parameter k is the size of a minimum proper interval deletion set D.

The algorithm for special weighted interval graphs

Consider a *special weighted interval graph* $G = (V, E)$ with parameter $\kappa \in \mathbb{N}$ and $|V| = n$, which is given along with a special interval representation $\mathcal{I}$ and a special vertex partition $V = A \cup B$. Recall from Definition 8 that A is an independent set and that $|B| \leq \kappa$. Let $w : V \to \mathbb{N}$ be the vertex weight function of G. For simplicity, we add to the set B an isolated dummy vertex v_0 such that $v_0 <_\sigma v_1$ and $w(v_0) = 0$ (cf. the first line of Algorithm 1). For every vertex $v \in B$, we define:

$$\xi_v = \begin{cases} l_u & \text{if } l_v \in I_u \text{ for some } u \in A, \\ l_v & \text{otherwise.} \end{cases} \tag{1}$$

Since A is an independent set by assumption, for every $v \in B$ there exists at most one $u \in A$ such that $l_v \in I_u$. Thus ξ_v is well-defined. Now we define the set $\Xi = \{\xi_v, l_v : v \in B\}$. Note that $|\Xi| = O(\kappa)$. Furthermore, for every $u, v \in V$, where $u \in N(v)$ and $u <_\sigma v$, we define the vertex $\pi_{u,v} = \max_\sigma\{\{u\} \cup \{w \in B \cap N(u) : u <_\sigma w <_\sigma v\}\}$. Note that, by definition, if $\pi_{u,v} \neq u$ then $\pi_{u,v} \in B$. Furthermore, due to the condition that $u \in N(v)$ in the definition of $\pi_{u,v}$, it follows that $u \in B$ or $v \in B$, since A is an independent set. That is, vertex $\pi_{u,v}$ is defined for at most $O(\kappa n)$ pairs of vertices u, v. Now we provide the crucial definition of the subgraphs $G_\xi(v_i)$ of G, for every $\xi \in \Xi$ and $i \in [n]$ such that $\xi < r_{v_i}$.

▶ **Definition 10.** Let $\xi \in \Xi$ and $i \in [n]$ such that $\xi < r_{v_i}$. We define the induced subgraph $G_\xi(v_i) = G[\{v \in V : \xi \leq l_v < r_v \leq r_{v_i}\}]$ of G which contains all vertices whose intervals are entirely contained between the points ξ and r_{v_i}.

▶ **Notation 1.** *Let $\xi \in \Xi$ and $i \in [n]$ such that $\xi < r_{v_i}$. Furthermore let $y \in V(G_\xi(v_i))$ such that $y \in N(v_i)$. We denote by $P_\xi(v_i, y)$ a* maximum-weight normal path *of $G_\xi(v_i)$, among those normal paths whose last vertex is y. For every path $P_\xi(v_i, y)$, we denote its weight $w(P_\xi(v_i, y))$ by $W_\xi(v_i, y)$.*

Note that, by Definition 10, if $l_{v_i} < \xi$ then the vertex v_i does not belong to the subgraph $G_\xi(v_i)$. Now we state three lemmas that are crucial for the correctness of Algorithm 1.

▶ **Lemma 11.** *Let $\xi \in \Xi$ and $i \in [n]$, where $\xi < r_{v_i}$, and let $y \in V(G_\xi(v_i))$ and $y \in N(v_i)$. If $l_y < l_{v_i}$ or $v_i \notin V(G_\xi(v_i))$, then $W_\xi(v_i, y) = W_\xi(\pi_{y,v_i}, y)$.*

▶ **Lemma 12.** *Let $\xi \in \Xi$ and $i \in [n]$, where $\xi < r_{v_i}$, and let $v_i \in V(G_\xi(v_i))$. Then $P_\xi(v_i, v_i) = (P_1, v_i)$, where $w(P_1) = \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < r_{v_i}\}$.*

▶ **Lemma 13.** *Let $\xi \in \Xi$ and $i \in [n]$, where $\xi < r_{v_i}$, and let $v_i, y \in V(G_\xi(v_i))$ and $y \in N(v_i)$. Let $P_\xi(v_i, y) = (P_1, v_i, P_2)$. If $P_2 \neq (y)$ then there exists some $\zeta \in \Xi$, where $l_{v_i} < \zeta \leq l_y$, such that*

$$w(P_1) = \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < \zeta\}, \tag{2}$$

$$w(P_2) = W_\zeta(\pi_{y,v_i}, y). \tag{3}$$

Otherwise, if $P_2 = (y)$ then $l_{v_i} < l_y$ and

$$w(P_1) = \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < l_y\}. \tag{4}$$

We are now ready to present Algorithm 1, which computes the maximum weight of a path in a given *special weighted* interval graph G. It is easy to check that Algorithm 1 can be slightly modified such that it returns the actual path P instead of its weight only.

Algorithm 1 Computing a maximum-weight path of a special interval graph

Input: A special weighted interval graph $G = (V, E)$ with parameter $\kappa \in \mathbb{N}$, along with the special interval representation $\mathcal{I}$ of G and the partition $V = A \cup B$, where $\sigma = (v_1, v_2, \ldots, v_n)$ is a right-endpoint ordering of V.

Output: The maximum weight of a path in G

1: Add an isolated dummy vertex v_0 with $w(v_0) = 0$ to set B, where $v_0 <_\sigma v_1$

2: **for** $i = 0$ to n **do**

3: **for** every $\xi \in \Xi$ where $\xi < r_{v_i}$ **do**

4: **if** $v_i \in V(G_\xi(v_i))$ **then** $W_\xi(v_i, v_i) \leftarrow w(v_i)$ {initialization}

5: **for** every $y \in V(G_\xi(v_i))$ where $y \in N(v_i)$ **do**

6: $W_\xi(v_i, y) \leftarrow W_\xi(\pi_{y,v_i}, y)$ {initialization}

7: **if** $v_i \in V(G_\xi(v_i))$ **then**

8: $W_1 \leftarrow \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < r_{v_i}\}$

9: $W_\xi(v_i, v_i) \leftarrow \max\{W_\xi(v_i, v_i), W_1 + w(v_i)\}$

10: **for** every $y \in V(G_\xi(v_i))$ where $y \in N(v_i)$ **do**

11: **if** $l_y < l_{v_i}$ or $v_i \notin V(G_\xi(v_i))$ **then**

12: $W_\xi(v_i, y) \leftarrow W_\xi(\pi_{y,v_i}, y)$

13: **else**

14: $W'_1 \leftarrow \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < l_y\}$

15: **for** every $\zeta \in \Xi$ with $l_{v_i} < \zeta \leq l_y$ **do**

16: $W_1 \leftarrow \max\{W_\xi(\pi_{x,v_i}, x) : x \in V(G_\xi(v_i)), l_{v_i} < r_x < \zeta\}$

17: $W_\xi(v_i, y) \leftarrow \max\{W_\xi(v_i, y), W_1 + w(v_i) + W_\zeta(\pi_{y,v_i}, y)\}$

18: $W_\xi(v_i, y) \leftarrow \max\{W_\xi(v_i, y), W'_1 + w(v_i) + w(y)\}$

19: **return** $\max\{W_{l_{v_0}}(v_i, v_i), W_{l_{v_0}}(v_i, y) : 1 \leq i \leq n,\ y <_\sigma v_i,\ y \in N(v_i)\}$

First we give a brief overview of the algorithm. Using dynamic programming, it computes a 3-dimensional table. In this table, for every point $\xi \in \Xi$, every index $i \in [n]$, and every vertex $y \in V(G_\xi(v_i))$, where $\xi < r_{v_i}$ and $y \in N(v_i)$, the entry $W_\xi(v_i, y)$ (resp. the entry $W_\xi(v_i, v_i)$) keeps the weight of a normal path in the subgraph $G_\xi(v_i)$ which is the largest among those normal paths whose last vertex is y (resp. v_i). Thus, since $w(v_0) = 0$ for the dummy isolated vertex v_0 (cf. line 1 of the algorithm), the maximum weight of a path in G will be eventually stored in one of the entries $\{W_{l_{v_0}}(v_i, v_i) : 1 \leq i \leq n\}$ or in one of the entries $\{W_{l_{v_0}}(v_i, y) : 1 \leq i \leq n,\ y <_\sigma v_i,\ y \in N(v_i)\}$, depending on whether the last vertex y of the desired maximum-weight path coincides with the rightmost vertex v_i of this path in the ordering σ (cf. line 19 of the algorithm).

Note that for every computed entry $W_\xi(v_i, y)$ the vertices v_i and y are adjacent, and thus $v_i \in B$ or $y \in B$, since A is an independent set. Thus, since $|B| = O(\kappa)$, there are at most $O(\kappa n)$ such eligible pairs of vertices v_i, y. Furthermore, since also $|\Xi| = O(\kappa)$, the computed 3-dimensional table stores at most $O(\kappa^2 n)$ entries $W_\xi(v_i, v_i)$ and $W_\xi(v_i, y)$. From the *for*-loops of lines 2-3 of the algorithm and from the obvious inductive hypothesis we may assume that during the $\{i, \xi\}$th iteration all previous values $W_{\xi'}(v_{i'}, v_{i'})$ and $W_{\xi'}(v_{i'}, y')$, where $i' < i$ or $\xi' < \xi$, have been correctly computed at a previous iteration.

In the initialization phase for a particular pair $\{i, \xi\}$ (cf. lines 4-6) the algorithm computes some initial values for $W_\xi(v_i, v_i)$ and $W_\xi(v_i, y)$. For a path with v_i as its last vertex, we are

only interested in the case where $v_i \in V(G_\xi(v_i))$; in this case we initialize $W_\xi(v_i, v_i) = w(v_i)$, cf. line 4. For a path with $y \neq v_i$ as its last vertex (cf. lines 5-6), we initialize $W_\xi(v_i, y) = W_\xi(\pi_{y,v_i}, y)$, since the path $P_\xi(\pi_{y,v_i}, y)$ is indeed a normal path of the graph $G_\xi(\pi_{y,v_i})$, which is an induced subgraph of $G_\xi(v_i)$.

For the induction step phase (cf. lines 7-18) the algorithm updates the initialized entries $W_\xi(v_i, v_i)$ and $W_\xi(v_i, y)$ according to Lemmas 11-13. To update the value $W_\xi(v_i, v_i)$ we only need to consider the case where $v_i \in V(G_\xi(v_i))$; in this case $W_\xi(v_i, v_i)$ is updated in lines 7-9 according to Lemma 12. The values of $W_\xi(v_i, y)$, where $y \neq v_i$, are updated in lines 10-18. In particular, in the case where $l_y < l_{v_i}$ or $v_i \notin V(G_\xi(v_i))$, the value of $W_\xi(v_i, y)$ is updated in lines 11-12 according to Lemma 11. Otherwise, $W_\xi(v_i, y)$ is updated in lines 14-18 according to Lemma 13.

▶ **Theorem 14.** *Let $G = (V, E)$ be a special weighted interval graph with n vertices and parameter κ. Then Algorithm 1 computes the maximum weight of a path P in G in $O(\kappa^3 n)$ time.*

The general algorithm

Here we present our *parameterized linear-time* algorithm for Longest Path on Interval Graphs, whose running time has a *polynomial dependency* on k.

▶ **Theorem 15.** *Let $G = (V, E)$ be an interval graph, where $|V| = n$ and $|E| = m$, and let k be the minimum size of a proper interval deletion set of G. Let $\mathcal{I}$ be an interval representation of G whose endpoints are sorted increasingly. Then:*

1. *a proper interval deletion set D, where $|D| \leq 4k$, can be computed in $O(n + m)$ time,*
2. *a semi-proper interval representation $\mathcal{I}'$ can be constructed in $O(n + m)$ time,*
3. *given D and $\mathcal{I}'$, a longest path of G can be computed in $O(k^9 n)$ time.*

5 Outlook

Our work heads at stimulating a general research program which systematically exploits the concept of fixed-parameter tractability for polynomially solvable problems. For several fundamental and widely known problems, the time complexity of the currently fastest algorithms are upper-bounded by polynomials of large degrees. One of the most prominent examples is arguably the celebrated polynomial-time recognition algorithm for *perfect graphs*, whose time complexity still remains $O(n^9)$ [14]. Apart from trying to improve the *worst-case* time complexity for such problems, which may be a very difficult (if not impossible) task, the complementary approach that we propose here is to try to detect the *parameter* that causes these high-degree polynomial-time algorithms and to separate the dependency of the time complexity from this parameter such that the dependency on the input size becomes as close to linear as possible. We believe that the "FPT inside P" field is very rich and offers plenty of research possibilities.

We conclude with two related topics that may lead to further interactions. First, we remark that in classical parameterized complexity analysis there is a growing awareness concerning the polynomial-time factors that often have been neglected [34]. Notably, there are some prominent fixed-parameter tractability results giving *linear-time* factors in the input size (but quite large exponential factors in the parameter); these include Bodlaender's famous algorithm for computing treewidth [8] and the more recent algorithm for computing the crossing number of a graph [24]. Interestingly, these papers emphasize "linear time" in their titles, instead of "fixed-parameter tractability". In this spirit, our result for Longest Path

in Interval graphs is a "linear-time" algorithm where the dependency on the parameter is not exponential [8, 24] but *polynomial*.

Second, polynomial-time solvability and the corresponding lower bounds have been of long-standing interest, e.g., it is believed that the famous 3SUM problem is only solvable in quadratic time and this conjecture has been employed for proving relative lower bounds for other problems [19]. Very recently, there was a significant push for this research direction with many new relative lower bounds [2, 1, 9]. The "FPT inside P" approach might help in "breaking" this nonlinear relative lower bounds by introducing useful parameterizations and striving for PL-FPT results, whenever possible; some very recent results in this direction concern the problems of computing the diameter and the radius in a graph [3].

References

1 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1681–1697, 2015.

2 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 434–443, 2014.

3 Amir Abboud, Virginia Vassilevska Williams, and Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2016. To appear.

4 Ittai Abraham, Amos Fiat, Andrew V. Goldberg, and Renato Fonseca F. Werneck. Highway dimension, shortest paths, and provably efficient algorithms. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 782–793, 2010.

5 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. of ACM*, 42(4):844–856, 1995.

6 Amihood Amir, Moshe Lewenstein, and Ely Porat. Faster algorithms for string matching with k mismatches. *Journal of Algorithms*, 50(2):257–275, 2004.

7 Alan A. Bertossi. Finding Hamiltonian circuits in proper interval graphs. *Information Processing Letters*, 17(2):97–101, 1983.

8 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. on Comp.*, 25(6):1305–1317, 1996.

9 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 661–670, 2014.

10 Sergio Cabello and Christian Knauer. Algorithms for graphs of bounded treewidth via orthogonal range searching. *Computational Geometry*, 42(9):815–824, 2009.

11 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

12 Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Homology flows, cohomology cuts. *SIAM J. on Comput.*, 41(6):1605–1634, 2012.

13 Jianer Chen, Songjian Lu, Sing-Hoi Sze, and Fenghui Zhang. Improved algorithms for path, matching, and packing problems. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 298–307, 2007.

14 Maria Chudnovsky, Gérard Cornuéjols, Xinming Liu, Paul Seymour, and Kristina Vušković. Recognizing Berge graphs. *Combinatorica*, 25(2):143–186, 2005.

15 Peter Damaschke. Paths in interval graphs and circular arc graphs. *Discrete Mathematics*, 112(1-3):49–64, 1993.

16 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

17 Michael R. Fellows, Bart M. P. Jansen, and Frances A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013.

18 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

19 Anka Gajentaan and Mark H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Computational Geometry*, 5:165–185, 1995.

20 Jiong Guo, Falk Hüffner, and Rolf Niedermeier. A structural view on parameterizing problems: Distance from triviality. In *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC)*, pages 162–173, 2004.

21 Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998.

22 Jan M. Hochstein and Karsten Weihe. Maximum s-t-flow with k crossings in $O(k^3 n \log n)$ time. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 843–847, 2007.

23 K. Ioannidou, George B. Mertzios, and Stavros D. Nikolopoulos. The longest path problem has a polynomial solution on interval graphs. *Algorithmica*, 61(2):320–341, 2011.

24 Ken-ichi Kawarabayashi and Bruce A. Reed. Computing crossing number in linear time. In *Proceedings of the 39th ACM Symp. on Th. of Comp. (STOC)*, pages 382–390, 2007.

25 J. M. Keil. Finding Hamiltonian circuits in interval graphs. *Information Processing Letters*, 20:201–206, 1985.

26 Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith. Divide-and-color. In *Proceedings of the 32nd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 58–67, 2006.

27 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-$m \log n$ time solver for SDD linear systems. In *Proceedings of the IEEE 52nd Symposium on Foundations of Computer Science (FOCS)*, pages 590–598, 2011.

28 Nimrod Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984.

29 George B. Mertzios and Derek G. Corneil. A simple polynomial algorithm for the longest path problem on cocomparability graphs. *SIAM J. on Discr. Math.*, 26(3):940–963, 2012.

30 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, 2006.

31 James B. Orlin, Kamesh Madduri, K. Subramani, and Matthew D. Williamson. A faster algorithm for the single source shortest path problem with few distinct positive lengths. *Journal of Discrete Algorithms*, 8(2):189–198, 2010.

32 F. S. Roberts. Indifference graphs. In F. Harary, editor, *Proof Techniques in Graph Theory*, pages 139–146. Academic Press, New York, 1969.

33 R. Uehara and Y. Uno. On computing longest paths in small graph classes. *International Journal of Foundations of Computer Science*, 18(5):911–930, 2007.

34 René van Bevern. *Fixed-Parameter Linear-Time Algorithms for NP-hard Graph and Hypergraph Problems Arising in Industrial Applications*. PhD thesis, Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany, 2014.

35 Ryan Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6):315–318, 2009.