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

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

[1]Institute of Software Engineering and Theoretical Computer Science, Logic and Semantics, TU Berlin, Germany

[2]School of Engineering and Computing Sciences, Durham University, UK

[3]Institute of Software Engineering and Theoretical Computer Science, Algorithms and Complexity Theory, TU Berlin, Germany

Department of Computer Science
University of London

March 2017

Fixed-parameter algorithms

For many combinatorial problems the best known (exact) algorithms are too slow:

- exponential running times (for NP-hard problems)
- polynomials of high degree, e.g. $O(n^3)$, $O(n^4)$, ... (for problems in P)

Fixed-parameter algorithms

For many combinatorial problems the best known (exact) algorithms are too slow:

- exponential running times (for NP-hard problems)
- polynomials of high degree, e.g. $O(n^3)$, $O(n^4)$, ... (for problems in P)

The successful "FPT approach" for exact computation:

- identify an appropriate parameter k that "causes" large running times
- design algorithms that separate the dependency of the running time from the input size n and the parameter k

Fixed-parameter algorithms

For many combinatorial problems the best known (exact) algorithms are too slow:

- exponential running times (for NP-hard problems)
- polynomials of high degree, e.g. $O(n^3)$, $O(n^4)$, ... (for problems in P)

The successful "FPT approach" for exact computation:

- identify an appropriate parameter k that "causes" large running times
- design algorithms that separate the dependency of the running time from the input size n and the parameter k

More formally:

- a fixed-parameter algorithm solves a problem with input size n and parameter k in $f(k) \cdot n^{O(1)}$ time

$\Rightarrow$ whenever k is small, the algorithm is efficient for every input size n

Fixed-parameter algorithms

- Fixed-Parameter Tractability (FPT) is a flourishing field, see e.g.
  [Downey, Fellows, *Parameterized Complexity*, 1999]
  [Flum, Grohe, *Parameterized Complexity Theory*, 2006]
  [Niedermeier, *Invitation to Fixed-Parameter Algorithms*, 2006]
  [Downey, Fellows, *Fundamentals of Parameterized Complexity*, 2013]
  [Cygan, Fomin, Kowalik, Lokshtanov, Marx, Pilipczuk^2, Saurabh, *Parameterized Algorithms*, 2015]
- So far, FPT research focused on intractable (NP-hard) problems
  - where the function $f(k)$ is unavoidably exponential (assuming P$\neq$NP)

Fixed-parameter algorithms

- Fixed-Parameter Tractability (FPT) is a flourishing field, see e.g.
  [Downey, Fellows, *Parameterized Complexity*, 1999]
  [Flum, Grohe, *Parameterized Complexity Theory*, 2006]
  [Niedermeier, *Invitation to Fixed-Parameter Algorithms*, 2006]
  [Downey, Fellows, *Fundamentals of Parameterized Complexity*, 2013]
  [Cygan, Fomin, Kowalik, Lokshtanov, Marx, Pilipczuk^2, Saurabh, *Parameterized Algorithms*, 2015]
- So far, FPT research focused on intractable (NP-hard) problems
  - where the function $f(k)$ is unavoidably exponential (assuming P$\neq$NP)
- There is a growing awareness about the polynomial factors $n^{O(1)}$ (which were usually neglected), e.g.:
  - computing the treewidth: [Bodlaender, *SIAM J. on Computing*, 1996]
  - computing the crossing number: [Kawarabayashi, Reed, *STOC*, 2007]
  - problems from industrial applications: [van Bevern, *PhD Thesis*, 2014]
  - these works emphasize "linear time" in the title, instead of "FPT"

“FPT inside P”

- Although polynomially solvable problems are theoretically tractable:
  - often the best known algorithms are not efficient in practice, e.g.
  - Linear Programming on arbitrary instances (interior point algorithms)
  - Matrix Multiplication (currently in $O(n^{2.373})$ time)
  - Maximum Matching (in $O(m\sqrt{n})$ time worst-case)

"FPT inside P"

- Although polynomially solvable problems are theoretically tractable:
  - often the best known algorithms are not efficient in practice, e.g.
  - Linear Programming on arbitrary instances (interior point algorithms)
  - Matrix Multiplication (currently in $O(n^{2.373})$ time)
  - Maximum Matching (in $O(m\sqrt{n})$ time worst-case)
- In certain applications (e.g. when working with massive data sets):
  - even $O(n^2)$-time is considered inefficient
- Reducing the worst-case complexity:
  - significant improvements are often difficult (or impossible)

"FPT inside P"

- Although polynomially solvable problems are theoretically tractable:
  - often the best known algorithms are not efficient in practice, e.g.
  - Linear Programming on arbitrary instances (interior point algorithms)
  - Matrix Multiplication (currently in $O(n^{2.373})$ time)
  - Maximum Matching (in $O(m\sqrt{n})$ time worst-case)
- In certain applications (e.g. when working with massive data sets):
  - even $O(n^2)$-time is considered inefficient
- Reducing the worst-case complexity:
  - significant improvements are often difficult (or impossible)
- Towards reducing polynomial factors $n^{O(1)}$:
  - the "FPT approach" can help refining the complexity of problems in P
- Appropriate parameterizations of a problem within P:
  - can reveal what makes it "far from being solvable in linear time"
  - in the same spirit as classical FPT algorithms (why is it "far from P")

"FPT inside P"

Formally, given a problem Π with instance size n:

- for which there exists an $O(n^c)$-time algorithm

we aim at detecting an appropriate parameter k such that:

- there exists an $f(k) \cdot n^{c'}$-time algorithm where
  1. $c' < c$ and
  2. $f(k)$ depends only on k

“FPT inside P”

Formally, given a problem Π with instance size n:

- for which there exists an $O(n^c)$-time algorithm

we aim at detecting an appropriate parameter k such that:

- there exists an $f(k) \cdot n^{c'}$-time algorithm where
  1. $c' < c$ and
  2. $f(k)$ depends only on k

Definition (refinement of FPT)

For every polynomially bounded function $p(n)$, the class $\text{FPT}(p(n))$ contains the problems solvable in $f(k) \cdot p(n)$ time,
where $f(k)$ is an arbitrary (possibly exponential) function of k.

"FPT inside P"

Formally, given a problem Π with instance size n:

- for which there exists an $O(n^c)$-time algorithm

we aim at detecting an appropriate parameter k such that:

- there exists an $f(k) \cdot n^{c'}$-time algorithm where
  1. $c' < c$ and
  2. $f(k)$ depends only on k

Definition (refinement of FPT)

For every polynomially bounded function $p(n)$, the class $\text{FPT}(p(n))$ contains the problems solvable in $f(k) \cdot p(n)$ time, where $f(k)$ is an arbitrary (possibly exponential) function of k.

For a problem within P:

- it is possible that $f(k)$ can become polynomial on k
- in wide contrast to FPT algorithms for NP-hard problems!

"FPT inside P"

Motivated by this:

Definition (refinement of P)

For every polynomially bounded function $p(n)$, the class P-FPT$(p(n))$ *(Polynomial Fixed-Parameter Tractable)* contains the problems solvable in $O(k^t \cdot p(n))$ time for some constant $t \geq 1$, i.e. $f(k) = k^t$.

"FPT inside P"

Motivated by this:

Definition (refinement of P)

For every polynomially bounded function $p(n)$, the class P-FPT$(p(n))$ *(Polynomial Fixed-Parameter Tractable)* contains the problems solvable in $O(k^t \cdot p(n))$ time for some constant $t \geq 1$, i.e. $f(k) = k^t$.

For the case where $p(n) = n$, the class P-FPT(n) is called PL-FPT *(Polynomial-Linear Fixed-Parameter Tractable)*.

“FPT inside P”

Motivated by this:

Definition (refinement of P)

For every polynomially bounded function $p(n)$, the class P-FPT$(p(n))$ *(Polynomial Fixed-Parameter Tractable)* contains the problems solvable in $O(k^t \cdot p(n))$ time for some constant $t \geq 1$, i.e. $f(k) = k^t$.

For the case where $p(n) = n$, the class P-FPT(n) is called PL-FPT *(Polynomial-Linear Fixed-Parameter Tractable)*.

This “FPT inside P” theme:

- interesting research direction
- too little explored so far
- few known results, scattered around in the literature

"FPT inside P"

We propose three desirable algorithmic properties:

1. the running time should depend polynomially on the parameter k
   $\Rightarrow$ the problem is in P-FPT$(p(n))$, for some polynomial $p(n)$

2. when k is constant, the running time should be as close to linear as possible
   $\Rightarrow$ the problem is in PL-FPT, or at least in P-FPT$(p(n))$ where $p(n) \approx n$

3. the parameter value (or a good approximation) should be computable efficiently (preferably in linear time) for arbitrary parameter values

"FPT inside P"

We propose three desirable algorithmic properties:

1. the running time should depend polynomially on the parameter k
   $\Rightarrow$ the problem is in P-FPT$(p(n))$, for some polynomial $p(n)$

2. when k is constant, the running time should be as close to linear as possible
   $\Rightarrow$ the problem is in PL-FPT, or at least in P-FPT$(p(n))$ where $p(n) \approx n$

3. the parameter value (or a good approximation) should be computable efficiently (preferably in linear time) for arbitrary parameter values

The "FPT inside P" framework should be systematically studied:

- exploiting the rich toolbox of parameterized algorithm design
  - e.g. data reductions, kernelization, ...
- having these three properties as a "compass"

Related work

Shortest path problems

- Some polynomial algorithms can be "tuned" with respect to specific parameters:
  - classic Dijkstra's algorithm for shortest paths: $O(m + n \log n)$ time
  - can be adapted to: $O(m + n \log k)$ time, where k is the number of distinct edge weights
    [Orlin, Madduri, Subramani, Williamson, *J. of Discr. Alg.*, 2010]
    [Koutis, Miller, Peng, *FOCS*, 2011]
- In order to prove the efficiency of known heuristics for road networks:
  - the parameter highway dimension has been introduced
    [Abraham, Fiat, Goldberg, Werneck, *SODA*, 2010]
  - plain Dijkstra's algorithm is too slow in practice

Conclusion: Adopting a parameterized view may be of significant practical interest, even for quasi-linear algorithms

Related work

Maximum flow problems

- For graphs made planar by deleting k crossing edges:
  - maximum flow in $O(k^3 \cdot n \log n)$ time [Hochstein, Weihe, *SODA*, 2007]
  - an embedding and the k crossing edges are given in the input

  $\Rightarrow$ this violates Property 3 (no known good approximation of k)

- For graphs made planar by deleting k crossing edges:
  - maximum flow in $O(k^3 \cdot n \log n)$ time [Hochstein, Weihe, *SODA*, 2007]
  - an embedding and the k crossing edges are given in the input
  - $\Rightarrow$ this violates Property 3 (no known good approximation of k)

- For graphs with bounded genus g and sum of capacities C:
  - maximum flow in $O(g^8 \cdot n \log^2 n \; log^2 C)$ time [Chambers, Erickson, Nayyeri, *SIAM J. on Computing*, 2012]
  - an embedding and the genus g are given in the input
  - $\Rightarrow$ this violates Property 3 (no known good approximation of g)

Related work

Maximum flow problems

- For graphs made planar by deleting k crossing edges:
  - maximum flow in $O(k^3 \cdot n \log n)$ time [Hochstein, Weihe, *SODA*, 2007]
  - an embedding and the k crossing edges are given in the input
  - $\Rightarrow$ this violates Property 3 (no known good approximation of k)
- For graphs with bounded genus g and sum of capacities C:
  - maximum flow in $O(g^8 \cdot n \log^2 n \; log^2 C)$ time [Chambers, Erickson, Nayyeri, *SIAM J. on Computing*, 2012]
  - an embedding and the genus g are given in the input
  - $\Rightarrow$ this violates Property 3 (no known good approximation of g)
- Furthermore, when parameterized by the treewidth k:
  - multiterminal flow in linear time [Hagerup, Katajainen, Nishimura, Ragde, *J. Comp. & Syst. Sci*, 1998]
  - Wiener index in near-linear time [Cabello, Knauer, *Comp. Geom.*, 2009]
  - both with exponential dependency on k
  - $\Rightarrow$ this violates Property 1 (exponential $f(k)$)

Related work

Linear Programming

- Due to a famous result of Megiddo [Megiddo, *J. of the ACM*, 1984]:
  - Linear Programming in linear time for fixed dimension d (# variables)
  - the multiplicative factor is $f(d) = 2^{O(2^d)}$

  $\Rightarrow$ this violates Property 1 (exponential $f(k)$), but is still in P-FPT(n)

  $\Rightarrow$ no guarantee for practically efficient algorithms
  - can be seen as a precursor of "FPT inside P"

- This result can be used similarly to Lenstra's result for ILP [Lenstra, *Math. of Operations Research*, 1983]:
  - Integer Linear Programming in FPT time for fixed dimension d
  - huge multiplicative factor $f(d)$

  $\Rightarrow$ mainly used for classification within FPT

Related work
Stringology

- String Matching with k Mismatches:
  - "find in a length-n string all occurrences of a length-m pattern with at most k errors"
  - in $O(m^2 + nk^2)$ [Landau, Vishkin, *FOCS*, 1985]
  - in $O(m \log k + nk^2)$ [Landau, Vishkin, *J. Comp. & Syst. Sci*, 1988]
  - in $O(nk)$ [Landau, Vishkin, *J. of Algorithms*, 1989]
  - in $O(n\sqrt{k \log k})$ [Amir, Lewenstein, Porat, *J. of Algorithms*, 2004]
- All these algorithms are linear in n
  - as also in the extreme case $k = 0$ errors

Related work
Stringology

- String Matching with k Mismatches:
  - "find in a length-n string all occurrences of a length-m pattern with at most k errors"
  - in $O(m^2 + nk^2)$ [Landau, Vishkin, *FOCS*, 1985]
  - in $O(m \log k + nk^2)$ [Landau, Vishkin, *J. Comp. & Syst. Sci*, 1988]
  - in $O(nk)$ [Landau, Vishkin, *J. of Algorithms*, 1989]
  - in $O(n\sqrt{k \log k})$ [Amir, Lewenstein, Porat, *J. of Algorithms*, 2004]
- All these algorithms are linear in n
  - as also in the extreme case $k = 0$ errors
- The parameter k is directly defined by the problem itself (and given with the input)
- Our approach goes beyond that:
  - we try to detect the appropriate parameter that causes a high polynomial time complexity

Our results

1. A "proof of concept" example: kernelization of `Maximum Matching`
   - parameter k = solution size
   - there exists a kernel with $O(k^2)$ vertices and edges
   - it can be computed in $O(kn)$ time

$\Rightarrow$ total running time: $O(kn + k^3)$

$\Rightarrow$ `Maximum Matching` is in PL-FPT for parameter k

A kernelization algorithm similar to Buss's for Vertex Cover:

- parameter k = solution size

Reduction Rule 1

If $\deg(v) > 2(k-1)$ *for some* $v \in V(G)$ *then return* $(G \setminus \{v\}, k-1)$.

Safeness (idea): if $(G \setminus \{v\}, k-1)$ is a YES-instance, then adding v can always produce a matching of size $\geq k$

- in a matching of size $k-1$ in $G \setminus \{v\}$, there is always "one more edge" in G

Kernelization of `Maximum Matching`

An illustrative example

A kernelization algorithm similar to Buss's for `Vertex Cover`:

- parameter k = solution size

Reduction Rule 1

If $\deg(v) > 2(k-1)$ *for some* $v \in V(G)$ *then return* $(G \setminus \{v\}, k-1)$.

Safeness (idea): if $(G \setminus \{v\}, k-1)$ is a `YES`-instance, then adding v can always produce a matching of size $\geq k$

- in a matching of size $k-1$ in $G \setminus \{v\}$, there is always "one more edge" in G

Reduction Rule 2

If $\deg(v) = 0$ *for some* $v \in V(G)$ *then return* $(G \setminus \{v\}, k)$.

Safeness: trivial

Kernelization of Maximum Matching

An illustrative example

Iteratively apply Reduction Rule 1:

- in total $O(kn)$ time

$\Rightarrow$ $\deg(v) \leq 2(k-1)$ for every (remaining) vertex v

Iteratively apply Reduction Rule 2:

- again in total $O(kn)$ time

$\Rightarrow$ $1 \leq \deg(v) \leq 2(k-1)$ for every (remaining) vertex v

Kernelization of `Maximum Matching`

An illustrative example

Iterately apply Reduction Rule 1:

- in total $O(kn)$ time

$\Rightarrow$ $\deg(v) \leq 2(k-1)$ for every (remaining) vertex v

Iteratively apply Reduction Rule 2:

- again in total $O(kn)$ time

$\Rightarrow$ $1 \leq \deg(v) \leq 2(k-1)$ for every (remaining) vertex v

We can easily prove for the remaining graph G':

Lemma

$|V(G')|, |E(G')| \leq (2k-1) \cdot \mathbf{mm}(G')$.

where $\mathbf{mm}(G')$ = size of maximum matching in G'

Kernelization of `Maximum Matching`

An illustrative example

Putting things together:

- compute the reduced graph G' (by Red. Rules $1+2$)
  - in total $O(kn)$ time
- suppose we remove r vertices by Reduction Rule 1
  - if $r \geq k$ then stop and return `YES`
  - else $k' = k - r$
- if G' has more than $(k'-1)(2k'-1)$ vertices or edges
  - then stop and return `YES`
  - else apply the best known algorithm for `Matching` on G'

Kernelization of `Maximum Matching`

An illustrative example

Putting things together:

- compute the reduced graph G' (by Red. Rules $1 + 2$)
  - in total $O(kn)$ time
- suppose we remove r vertices by Reduction Rule 1
  - if $r \geq k$ then stop and return `YES`
  - else $k' = k - r$
- if G' has more than $(k' - 1)(2k' - 1)$ vertices or edges
  - then stop and return `YES`
  - else apply the best known algorithm for `Matching` on G'

The best known worst-case algorithm:

- in $O(m\sqrt{n}) = O(k^3)$ time [Micali, Vazirani, *FOCS*, 1980]

$\Rightarrow$ total running time: $O(kn + k^3)$ time

Our results

2. Main technical result: `Longest Path` on `Interval Graphs`
   - `Longest Path` is polynomially solvable in several "small" graph classes:
     - weighted trees, block graphs, ptolemaic graphs, cacti, threshold graphs [Uehara, Uno, *ISAAC*, 2004]

   and only in a few "non-trivial" graph classes:
     - interval graphs, cocomparability graphs, both in $O(n^4)$ time [Ioannidou, Mertzios, Nikolopoulos, *Algorithmica*, 2011] [Mertzios, Corneil, *SIAM J. on Discrete Mathematics*, 2012]

Our results

2. Main technical result: `Longest Path` on `Interval Graphs`
   - `Longest Path` is polynomially solvable in several "small" graph classes:
     - weighted trees, block graphs, ptolemaic graphs, cacti, threshold graphs [Uehara, Uno, *ISAAC*, 2004]

   and only in a few "non-trivial" graph classes:
     - interval graphs, cocomparability graphs, both in $O(n^4)$ time [Ioannidou, Mertzios, Nikolopoulos, *Algorithmica*, 2011] [Mertzios, Corneil, *SIAM J. on Discrete Mathematics*, 2012]
   - On proper interval graphs:
     - trivially solvable in linear time
     - connected $\Rightarrow$ Hamiltonian

$\Rightarrow$ parameter distance to triviality:
   - $k =$ proper interval (vertex) deletion number
   - k can be 4-approximated in $O(n+m)$ time

2. Main technical result: `Longest Path` on `Interval Graphs`
   - `Longest Path` is polynomially solvable in several "small" graph classes:
     - weighted trees, block graphs, ptolemaic graphs, cacti, threshold graphs [Uehara, Uno, *ISAAC*, 2004]

     and only in a few "non-trivial" graph classes:
     - interval graphs, cocomparability graphs, both in $O(n^4)$ time [Ioannidou, Mertzios, Nikolopoulos, *Algorithmica*, 2011] [Mertzios, Corneil, *SIAM J. on Discrete Mathematics*, 2012]
   - On proper interval graphs:
     - trivially solvable in linear time
     - connected $\Rightarrow$ Hamiltonian

   $\Rightarrow$ parameter distance to triviality:
     - $k =$ proper interval (vertex) deletion number
     - k can be 4-approximated in $O(n+m)$ time

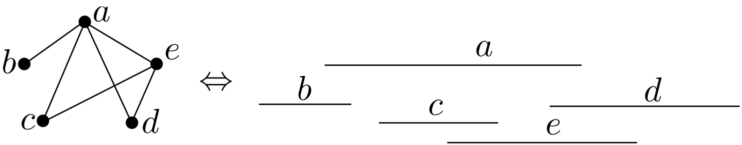
Our Algorithm: compute a longest path in $O(k^9 n)$ time

$\Rightarrow$ `Longest Path on Interval Graphs` is in PL-FPT for parameter k

Longest Path on Interval Graphs

Definition

A graph G is called an interval graph, if G is the intersection graph of a set of intervals on the real line.



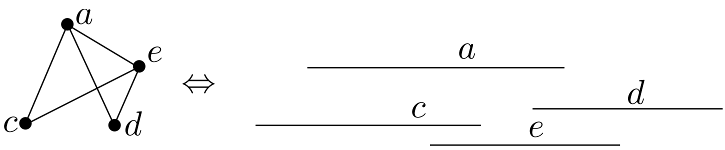
Longest Path on Interval Graphs

Definition

A graph G is called an interval graph, if G is the intersection graph of a set of intervals on the real line.

Definition

An interval graph G is a proper interval graph, if there exists an interval representation of G where no interval is properly included in another one.



Longest Path on Interval Graphs

Definition

A graph G is called an interval graph, if G is the intersection graph of a set of intervals on the real line.

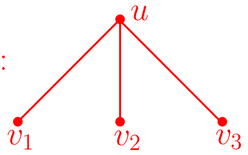
Definition

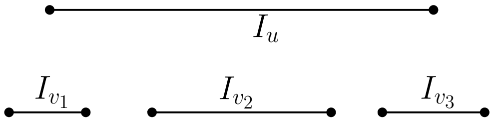
An interval graph G is a proper interval graph, if there exists an interval representation of G where no interval is properly included in another one.

Theorem (Roberts, 1969)

An interval graph G is a proper interval graph $\iff$ G does not include any claw $K_{1,3}$ as induced subgraph.

$K_{1,3}$:





Proper interval deletion set

We take as input:

- an interval representation of G
- G has n vertices and m edges
- the endpoints of the intervals are sorted increasingly

Proper interval deletion set

We take as input:

- an interval representation of G
- G has n vertices and m edges
- the endpoints of the intervals are sorted increasingly

Computation of a minimum proper interval deletion set D:

- Cai's algorithm (one forbidden subgraph): in $O(4^{|D|}poly(n))$ time [Cai, *Information Processing Letters*, 1996]
- polynomial time exact computation: Open problem!

Proper interval deletion set

We take as input:

- an interval representation of G
- G has n vertices and m edges
- the endpoints of the intervals are sorted increasingly

Computation of a minimum proper interval deletion set D:

- Cai's algorithm (one forbidden subgraph): in $O(4^{|D|}poly(n))$ time [Cai, *Information Processing Letters*, 1996]
- polynomial time exact computation: Open problem!

We compute a 4-approximation of $|D|$ in $O(n+m)$ time:

- scan from left to right in the interval representation
- detect a claw $K_{1,3}$
- remove all 4 vertices of the claw
- iterate

Longest Path on Interval Graphs

Normal paths in interval graphs

- Our proofs are based on the notion of normal paths in interval graphs.
  [Ioannidou, Mertzios, Nikolopoulos, *Algorithmica*, 2011]
  (a.k.a. straight paths: [Damaschke, *Discr. Math*, 1993])
- We assume the right-end ordering π of the vertices, i.e. after sorting the intervals according to the right endpoints.

- Our proofs are based on the notion of normal paths in interval graphs.
  [Ioannidou, Mertzios, Nikolopoulos, *Algorithmica*, 2011]
  (a.k.a. straight paths: [Damaschke, *Discr. Math*, 1993])
- We assume the right-end ordering π of the vertices, i.e. after sorting the intervals according to the right endpoints.

Definition

The path $P = (v_1, v_2, \ldots, v_k)$ of an interval graph G is a normal path, if:

- v_1 is the leftmost vertex of $V(P)$ in π, and
- v_i is the leftmost vertex of $N(v_{i-1}) \cap \{v_i, v_{i+1}, \ldots, v_k\}$ in π, for every $i \geq 2$.

- Our proofs are based on the notion of normal paths in interval graphs.
  [Ioannidou, Mertzios, Nikolopoulos, *Algorithmica*, 2011]
  (a.k.a. straight paths: [Damaschke, *Discr. Math*, 1993])
- We assume the right-end ordering π of the vertices, i.e. after sorting the intervals according to the right endpoints.

Definition

The path $P = (v_1, v_2, \ldots, v_k)$ of an interval graph G is a normal path, if:

- v_1 is the leftmost vertex of $V(P)$ in π, and
- v_i is the leftmost vertex of $N(v_{i-1}) \cap \{v_i, v_{i+1}, \ldots, v_k\}$ in π, for every $i \geq 2$.

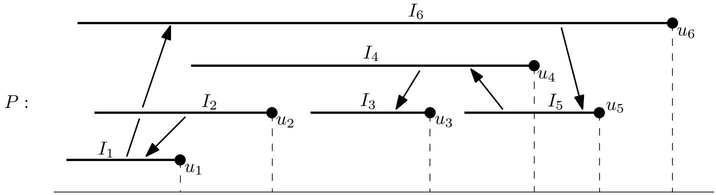
Lemma

For every path P in an interval graph G, there exists a normal path P' of G, such that $V(P') = V(P)$.

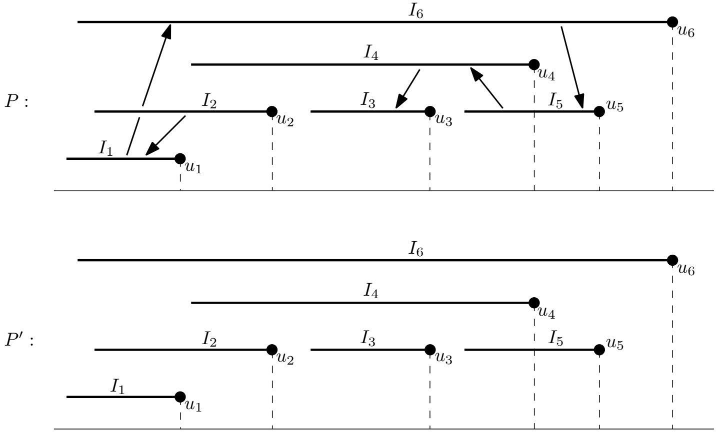
Longest Path on Interval Graphs

Normal paths in interval graphs

Example: path $P = (u_2, u_1, u_6, u_5, u_4, u_3)$

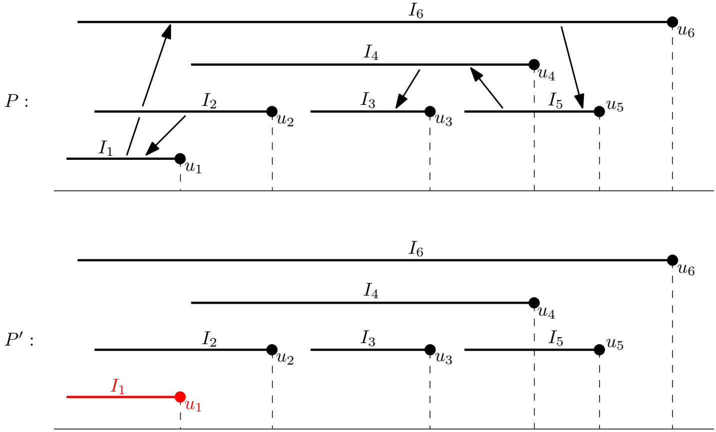


Example: path $P = (u_2, u_1, u_6, u_5, u_4, u_3)$



Normal path: $P' = (\quad , \quad , \quad , \quad , \quad , \quad)$

Example: path $P = (u_2, u_1, u_6, u_5, u_4, u_3)$

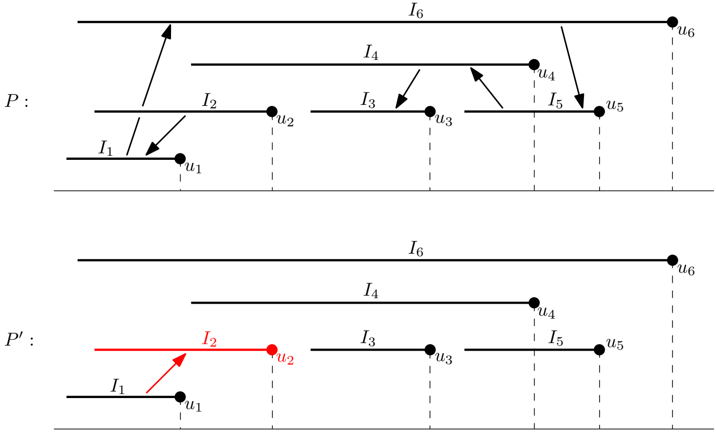


Normal path: $P' = (u_1, \quad , \quad , \quad , \quad , \quad)$

Longest Path on Interval Graphs

Normal paths in interval graphs

Example: path $P = (u_2, u_1, u_6, u_5, u_4, u_3)$

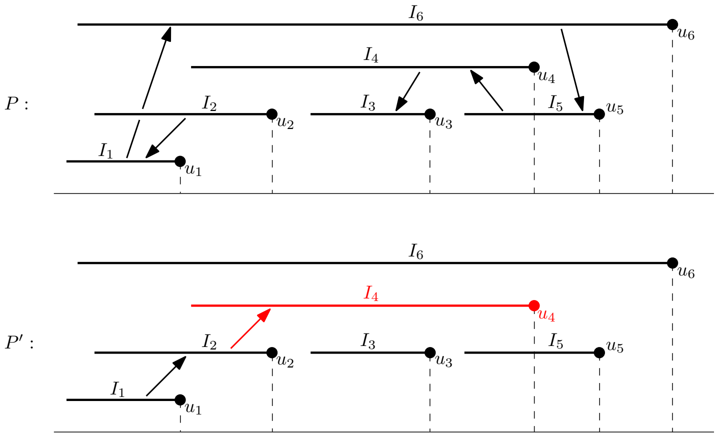


Normal path: $P' = (u_1, u_2, \quad , \quad , \quad , \quad)$

Longest Path on Interval Graphs

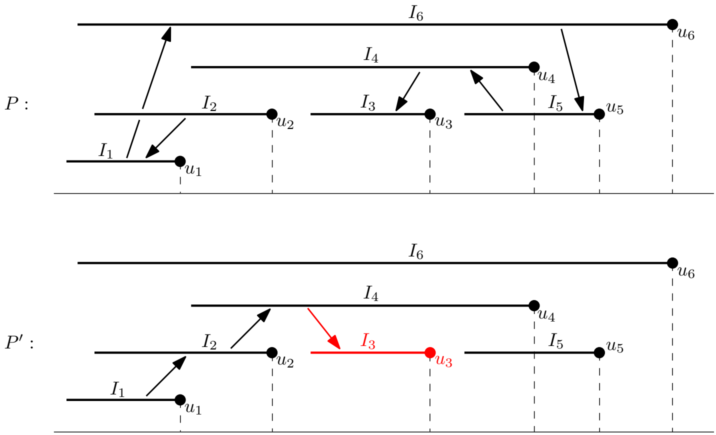
Normal paths in interval graphs

Example: path $P = (u_2, u_1, u_6, u_5, u_4, u_3)$



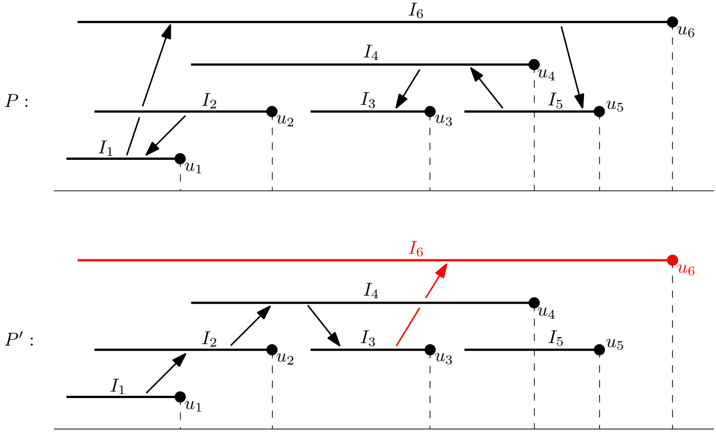
Normal path: $P' = (u_1, u_2, u_4, \quad , \quad , \quad)$

Example: path $P = (u_2, u_1, u_6, u_5, u_4, u_3)$



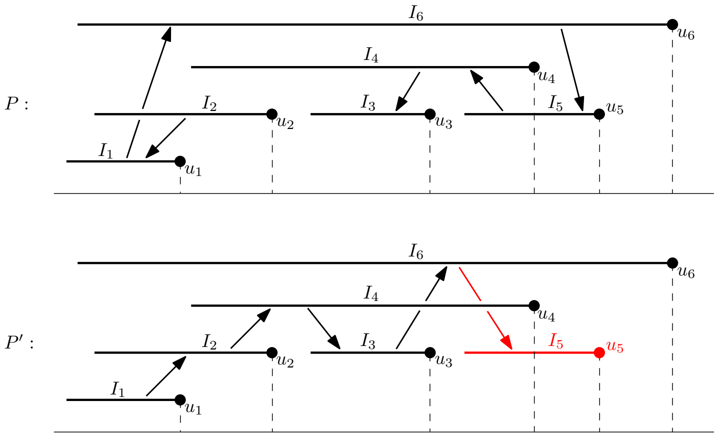
Normal path: $P' = (u_1, u_2, u_4, u_3, \quad , \quad)$

Example: path $P = (u_2, u_1, u_6, u_5, u_4, u_3)$



Normal path: $P' = (u_1, u_2, u_4, u_3, {\color{red}u_6}, \quad)$

Example: path $P = (u_2, u_1, u_6, u_5, u_4, u_3)$



Normal path: $P' = (u_1, u_2, u_4, u_3, u_6, u_5)$

Longest Path on Interval Graphs: Algorithm sketch

Given a proper interval deletion set D of G, where $|D| = k$:

1. partition $G \setminus D$ into:
   - a collection of "reducible" sets and
   - a collection of "weakly reducible" sets

Longest Path on Interval Graphs: Algorithm sketch

Given a proper interval deletion set D of G, where $|D| = k$:

1. partition $G \setminus D$ into:
   - a collection of "reducible" sets and
   - a collection of "weakly reducible" sets
2. exhaustively apply a data reduction rule
   - replace every reducible set with one weighted interval
   - $O(n)$ such new intervals

Longest Path on Interval Graphs: Algorithm sketch

Given a proper interval deletion set D of G, where $|D| = k$:

1. partition $G \setminus D$ into:
   - a collection of "reducible" sets and
   - a collection of "weakly reducible" sets
2. exhaustively apply a data reduction rule
   - replace every reducible set with one weighted interval
   - $O(n)$ such new intervals
3. exhaustively apply a second data reduction rule
   - replace every weakly reducible set with $O(k)$ weighted intervals
   - $O(k^3)$ such new intervals

Longest Path on Interval Graphs: Algorithm sketch

Given a proper interval deletion set D of G, where $|D| = k$:

1. partition $G \setminus D$ into:
   - a collection of "reducible" sets and
   - a collection of "weakly reducible" sets
2. exhaustively apply a data reduction rule
   - replace every reducible set with one weighted interval
   - $O(n)$ such new intervals
3. exhaustively apply a second data reduction rule
   - replace every weakly reducible set with $O(k)$ weighted intervals
   - $O(k^3)$ such new intervals
4. the resulting interval graph $\widehat{G}$ is weighted
   - $\widehat{G}$ is a "special weighted interval graph with parameter κ"
   - where $\kappa = O(k^3)$

Longest Path on Interval Graphs: Algorithm sketch

Given a proper interval deletion set D of G, where $|D| = k$:

1. partition $G \setminus D$ into:
   - a collection of "reducible" sets and
   - a collection of "weakly reducible" sets
2. exhaustively apply a data reduction rule
   - replace every reducible set with one weighted interval
   - $O(n)$ such new intervals
3. exhaustively apply a second data reduction rule
   - replace every weakly reducible set with $O(k)$ weighted intervals
   - $O(k^3)$ such new intervals
4. the resulting interval graph $\widehat{G}$ is weighted
   - $\widehat{G}$ is a "special weighted interval graph with parameter κ"
   - where $\kappa = O(k^3)$
5. dynamic programming algorithm on $\widehat{G}$
   - compute in $O(\kappa^3 n) = O(k^9 n)$ time a max. weight path in $\widehat{G}$
   - this corresponds to a longest path of G

Main properties of a reducible set S:

- for any longest path P of G, either $S \subseteq V(P)$ or $S \cap V(P) = \emptyset$
- if $S \subseteq V(P)$ for a longest normal path P,
  then the vertices of S appear consecutively in P

Main properties of a reducible set S:

- for any longest path P of G, either $S \subseteq V(P)$ or $S \cap V(P) = \emptyset$
- if $S \subseteq V(P)$ for a longest normal path P,
  then the vertices of S appear consecutively in P

$\Rightarrow$ we can replace S by one vertex of weight $|S|$

Therefore:

Reduction Rule (first data reduction)

Let $\mathcal{I}$ be an interval representation of G. If S is a reducible set, where $S \cap D = \emptyset$, then replace in $\mathcal{I}$ the intervals $\{I_v : v \in S\}$ with the single interval $I_S = \mathbf{span}(S)$ which has weight $|S|$.

Therefore:

Reduction Rule (first data reduction)

Let $\mathcal{I}$ be an interval representation of G. If S is a reducible set, where $S \cap D = \emptyset$, then replace in $\mathcal{I}$ the intervals $\{I_v : v \in S\}$ with the single interval $I_S = \mathbf{span}(S)$ which has weight $|S|$.

Remarks:

- there can be $O(n)$ reducible sets
- the new (weighted) vertices form an independent set
- all reducible sets can be replaced in total $O(n)$ time
  - by a left-to-right sweep in the interval representation

Longest Path on Interval Graphs

The second data reduction

- Recall: D is a proper interval deletion set of G, where $|D| = k$.

Main properties of a weakly reducible set S:

- for any longest path P of G, either $S \subseteq V(P)$ or $S \cap V(P) = \emptyset$
- if $S \subseteq V(P)$ for a longest normal path P, then the vertices of S in P are interrupted at most $k+3$ times by vertices outside S
  - otherwise $G \setminus D$ has a claw, contradiction!

- Recall: D is a proper interval deletion set of G, where $|D| = k$.

Main properties of a weakly reducible set S:

- for any longest path P of G, either $S \subseteq V(P)$ or $S \cap V(P) = \emptyset$
- if $S \subseteq V(P)$ for a longest normal path P, then the vertices of S in P are interrupted at most $k+3$ times by vertices outside S
  - otherwise $G \setminus D$ has a claw, contradiction!

$\Rightarrow$ we can replace S by $\min\{|S|, k+4\}$ vertices, each of equal weight

Reduction Rule (second data reduction)

Let $\mathcal{I}$ be an interval representation of G. If S is a weakly reducible set, where $S \cap D = \emptyset$, then replace in $\mathcal{I}$ the intervals $\{I_v : v \in S\}$ with $\min\{|S|, k+4\}$ *copies of the interval* $I_S = \mathbf{span}(S)$, *each with weight*

$$\frac{1}{\min\{|S_i|, k+4\}} \cdot \sum_{u \in S} w(u)$$

Reduction Rule (second data reduction)

Let $\mathcal{I}$ be an interval representation of G. If S is a weakly reducible set, where $S \cap D = \emptyset$, then replace in $\mathcal{I}$ the intervals $\{I_v : v \in S\}$ with $\min\{|S|, k+4\}$ copies of the interval $I_S = \textbf{span}(S)$, each with weight

$$\frac{1}{\min\{|S_i|, k+4\}} \cdot \sum_{u \in S} w(u)$$

Remarks:

- every weakly reducible set S corresponds to one pair of endpoints of the intervals in D

$\Rightarrow$ in total $O(k^2)$ weakly reducible sets

$\Rightarrow$ in total $O(k^3)$ new (weighted) vertices

- all weakly reducible sets can be replaced in total $O(k^2 n)$ time
  - by $O(k^2)$ left-to-right sweeps in the interval representation

Definition (special weighted interval graph with parameter κ)

Let $G = (V, E)$ be a weighted interval graph, where $V = A \dot{\cup} B$, let $\mathcal{I}$ be an interval representation of G, and let $\kappa \in \mathbb{N}$ such that:

1. A is an independent set,
2. $|B| \leq \kappa$,
3. for every $v \in A$ and every $u \in V \setminus \{v\}$, we have $I_u \nsubseteq I_v$ in $\mathcal{I}$.

Then G is a special weighted interval graph with parameter κ.

Definition (special weighted interval graph with parameter κ)

Let $G = (V, E)$ be a weighted interval graph, where $V = A \dot{\cup} B$, let $\mathcal{I}$ be an interval representation of G, and let $\kappa \in \mathbb{N}$ such that:

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

Then G is a special weighted interval graph with parameter κ.

We can prove:

Theorem

After replacing all reducible and weakly reducible sets, the resulting graph $\widehat{G}$ is a special weighted interval graph with parameter $\kappa = O(k^3)$.

(given a proper interval deletion set D of G, where $|D| = k$)

Definition (special weighted interval graph with parameter κ)

Let $G = (V, E)$ be a weighted interval graph, where $V = A \dot{\cup} B$, let $\mathcal{I}$ be an interval representation of G, and let $\kappa \in \mathbb{N}$ such that:

1. A is an independent set,
2. $|B| \leq \kappa$,
3. for every $v \in A$ and every $u \in V \setminus \{v\}$, we have $I_u \nsubseteq I_v$ in $\mathcal{I}$.

Then G is a special weighted interval graph with parameter κ.

Remarks:

- A are the vertices that replaced all reducible sets $\Rightarrow |A| = O(n)$
- B contains the vertices of D and the $O(k^3)$ replacements of the weakly reducible sets $\Rightarrow |B| = k + O(k^3) = O(k^3)$

$\Rightarrow \kappa = O(k^3)$

Definition (special weighted interval graph with parameter κ)

Let $G = (V, E)$ be a weighted interval graph, where $V = A \dot{\cup} B$, let $\mathcal{I}$ be an interval representation of G, and let $\kappa \in \mathbb{N}$ such that:

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

Then G is a special weighted interval graph with parameter κ.

Remarks:

- $\widehat{G} = (\widehat{V}, \widehat{E})$ can be computed from G in $O(k^2 n)$ time
- maximum weight path in $\widehat{G} \Longleftrightarrow$ longest path in G
- for every edge $uv \in \widehat{E}$: $u \in B$ or $v \in B$, since A is independent

$\Rightarrow |\widehat{E}| = O(\kappa n) = O(k^3 n)$

Definition (special weighted interval graph with parameter κ)

Let $G = (V, E)$ be a weighted interval graph, where $V = A\dot{\cup}B$, let $\mathcal{I}$ be an interval representation of G, and let $\kappa \in \mathbb{N}$ such that:

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

Then G is a special weighted interval graph with parameter κ.

Theorem

A maximum weight path in the special weighted interval graph $\widehat{G}$ can be computed in $O(\kappa^3 n) = O(k^9 n)$ time.

Conclusions & Outlook

- "FPT inside P" offers an alternative way to deal with problems in P:
  - $f(k)$ can possibly become polynomial
  - a nice interplay with fast approximation algorithms, providing suitable parameters
  - one can aim at reducing "slow" polynomial running times (e.g. $O(n^3)$ or higher)
  - but also $O(n^2)$ (or less) for more practical applications
- `Longest Path on Interval Graphs`:
  - Can we significantly improve the running time of $O(k^9 n)$?

Conclusions & Outlook

Follow-up work form other groups:

1. [Fomin, Lokshtanov, Pilipczuk, Saurabh, Wrochna, *arXiv*, 2015]

   - $O(t^3 n \log n)$-time randomized algorithm to compute the cardinality of a maximum matching,
   - $O(t^4 n \log^2 n)$-time randomized algorithm to construct a maximum matching,

   where t is the treewidth of the graph.

2. [Abboud, Vassilevska Williams, Wang, *SODA*, 2016]

   - $t^{O(t)} n^{1+o(1)}$-time algorithms to compute the diameter & radius,
   - no $2^{o(t)} n^{2-\varepsilon}$-time algorithm for even a $(\frac{3}{2} - \delta)$-approximation of the diameter or radius, subject to plausible (polynomial-time) complexity assumptions,

   where t is the treewidth of the graph.

Conclusions & Outlook

- Exploit the rich toolbox of “classical” FPT algorithms:
  - data reductions
  - kernelization
  - ...
- Lower bounds subject to established complexity conjectures
  - 3SUM item SETH
  - Boolean Matrix Multiplication
  - ...
- “FPT inside P” for big data / streaming
- Implementation / experiments of newly developed algorithms

Thank you for your attention!