KERNELIZATION

Polynomial time preprocessing - data reduction

Christophe Paul

Journée Algo & Calcul - NUMEV
11 Janvier 2012

What does computational theory tell us ?

- SAT is NP-Complete [Cook's Theorem, 1971]
  + Karp's list of 21 NP-Complete problems [1972]

  - Don't expect efficient exact algorithms for a number of problems !

What does computational theory tell us ?

- SAT is NP-Complete [Cook's Theorem, 1971]
  + Karp's list of 21 NP-Complete problems [1972]

  - Don't expect efficient exact algorithms for a number of problems !

- PCP Theorem (Gödel Prize 2001 to Arora, Feige, Goldwasser, Lund, Lovasz, Motwani Safra, Sudan and Szegedy)

  (Independent set cannot be approximated)

  - Don't expect efficient approximation algorithms for a number of problems !

What does computational theory tell us ?

- SAT is NP-Complete [Cook's Theorem, 1971]
  + Karp's list of 21 NP-Complete problems [1972]

  - Don't expect efficient exact algorithms for a number of problems !

- PCP Theorem (Gödel Prize 2001 to Arora, Feige, Goldwasser, Lund, Lovasz, Motwani Safra, Sudan and Szegedy)

  (Independent set cannot be approximated)

  - Don't expect efficient approximation algorithms for a number of problems !

- and so on ...

What does computational theory tell us ?

[. . . While theoretical work on models of computation and methods for analyzing algorithms has had enormous payoffs, we are not done. In many situations, simple algorithms do well. We don't understand why!

In **Challenges for theory of computing**: Report for an NSF-sponsored workshop on research in theoretical computer science.
Condon, Edelsbrunner, Emerson, Fortnow, Haber, Karp, Leivant, Lipton, Lynch, Parberry, Papadimitriou, Rabin, Rosenberg, Royer, Savage, Selman, Smith, Tardos, and Vitter.

Available at http://www.cs.buffalo.edu/selman/report/, 1999.

What does computational theory tell us ?

[. . . While theoretical work on models of computation and methods for analyzing algorithms has had enormous payoffs, we are not done. In many situations, simple algorithms do well. We don't understand why!

Developing means for predicting the performance of algorithms and heuristics on real data and on real computers is a grand challenge in algorithms. . .]

In **Challenges for theory of computing**: Report for an NSF-sponsored workshop on research in theoretical computer science.
Condon, Edelsbrunner, Emerson, Fortnow, Haber, Karp, Leivant, Lipton, Lynch, Parberry, Papadimitriou, Rabin, Rosenberg, Royer, Savage, Selman, Smith, Tardos, and Vitter.

Available at http://www.cs.buffalo.edu/selman/report/, 1999.

Effective Heuristics for NP-Hard Problems by R. M. Karp

Presented at the Michael Rabin Celebration, August 2011

In many practical situations heuristic algorithms reliably give satisfactory solutions to real-life instances of optimization problems, despite evidence from computational complexity theory that the problems are intractable in general. Our long-term goal is to contribute to an understanding of this seeming contradiction, and to put the construction of heuristic algorithms on a firmer footing.

Effective Heuristics for NP-Hard Problems by R. M. Karp

Presented at the Michael Rabin Celebration, August 2011

In many practical situations heuristic algorithms reliably give satisfactory solutions to real-life instances of optimization problems, despite evidence from computational complexity theory that the problems are intractable in general. Our long-term goal is to contribute to an understanding of this seeming contradiction, and to put the construction of heuristic algorithms on a firmer footing.

We begin by describing the evolution of successful heuristic algorithms for two problems arising in computational biology:
(1) The Colorful Subgraph Problem. [. . .]
(2) Global Genome Alignment. [. . .]

We then discuss a general approach for tuning the parameters and design choices within a given heuristic algorithmic strategy, assuming the availability of a training set of typical problem instances.

Polynomial time preprocessing

"*It has become clear, however, that far from being trivial and uninteresting, pre-processing has unexpected practical power for real world input distributions, and is mathematically a much deeper subject than has generally been understood.*"

M. Fellows. The lost continent of polynomial time: Preprocessing and kernelization, in IWPEC 2006

Polynomial time preprocessing

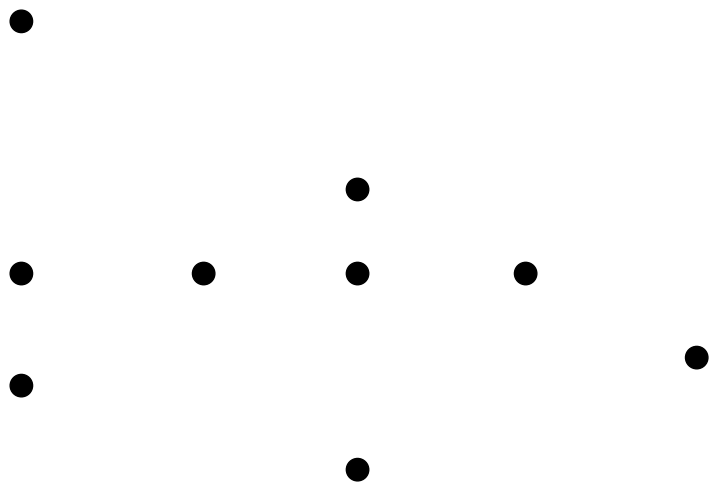
"*It has become clear, however, that far from being trivial and uninteresting, pre-processing has unexpected practical power for real world input distributions, and is mathematically a much deeper subject than has generally been understood.*"

M. Fellows. The lost continent of polynomial time: Preprocessing and kernelization, in IWPEC 2006

What can we expect from polynomial time preprocessing ?

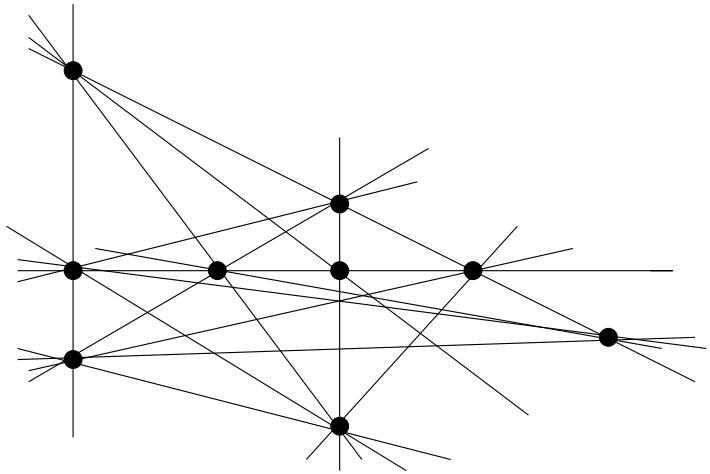
- How small can a data-set be skrinked, while preserving the existence of a solution ?
- How to measure the efficiency of preprocessing ?
- Does every NP-Hard problem enjoy an efficient polynomial time preprocessing ?
- Can we provide a mathematical framework to explain the power and limitation of preprocessing ?

Kernelization (an example)



Is there a set of k lines covering the set S of points ?

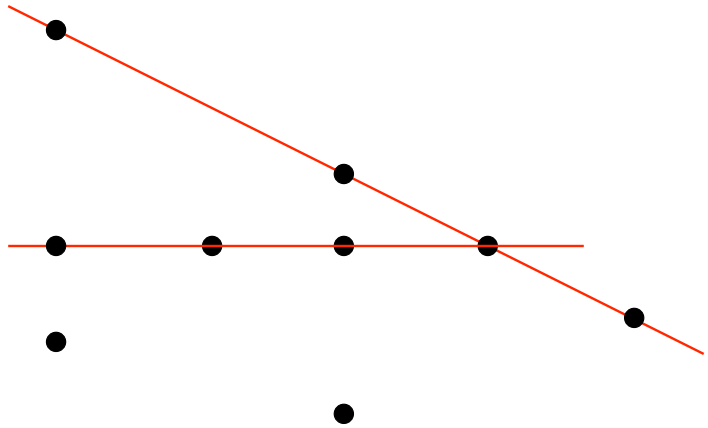
Kernelization (an example)



Is there a set of k lines covering the set S of points ?

Observation 1: We can restrict our attention to lines generated by the pairs of points of S

Kernelization (an example)

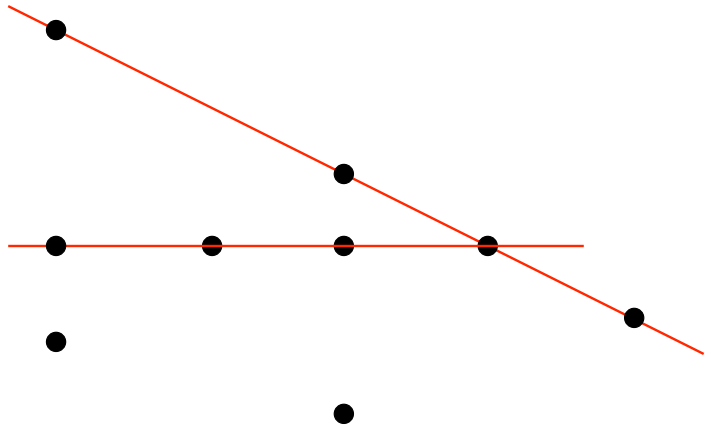


Is there a set of k lines covering the set S of points ?

Observation 2: If a line contains at least $k+1$ points, then any solution contains that line (e.g. $k = 3$)

$\Rightarrow$ [Degree rule] remove such a line and decrease k by 1

Kernelization (an example)



Is there a set of k lines covering the set S of points ?

Observation 2: If a line contains at least $k+1$ points, then any solution contains that line (e.g. $k = 3$)

$\Rightarrow$ [Degree rule] remove such a line and decrease k by 1

$\Rightarrow$ the reduced instance contains at most k^2 points.

Kernelization (definition)

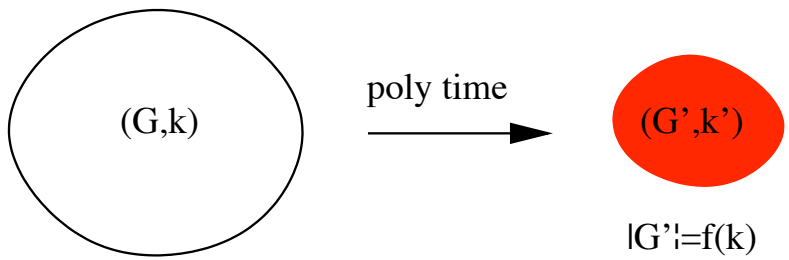
How to measure the size of the reduced instance ?

- need of a parameter independent from the data-size
- the parameter may express some structure of the instance (e.g. treewidth)
- the smaller the parameter is the better it is...
  (to be discussed later)

Kernelization (definition)

How to measure the size of the reduced instance ?

- need of a parameter independent from the data-size
- the parameter may express some structure of the instance (e.g. treewidth)
- the smaller the parameter is the better it is...
  (to be discussed later)



Given a parameterized instance $(\mathcal{I}, k)$ of a problem, a kernelization algorithm computes in polytime an equivalent instance $(\mathcal{I}', k')$ st.

$$k' = f(k) \qquad \text{and} \qquad |\mathcal{I}'| \leqslant g(k)$$

Kernelization (objective)

- a small kernel size $\longrightarrow$ linear . . . cubic . . . exponential ?

Kernelization (objective)

- a small kernel size $\longrightarrow$ linear . . . cubic . . . exponential ?
- a small parameter $\longrightarrow$ hierarchy of parameters

Kernelization (objective)

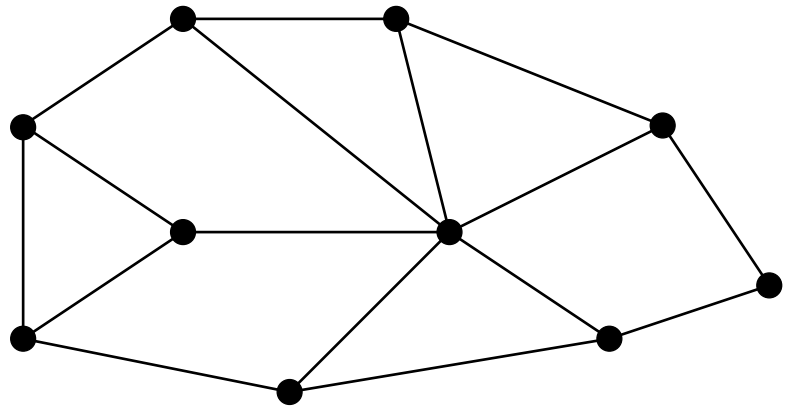
- a small kernel size $\longrightarrow$ linear . . . cubic . . . exponential ?
- a small parameter $\longrightarrow$ hierarchy of parameters
- efficient algorithm $\longrightarrow$ linear time . . . cubic time

Kernelization (objective)

- a small kernel size $\longrightarrow$ linear ... cubic ... exponential ?
- a small parameter $\longrightarrow$ hierarchy of parameters
- efficient algorithm $\longrightarrow$ linear time ... cubic time
- equivalence of the solutions $\longrightarrow$ at least a certain guarantee

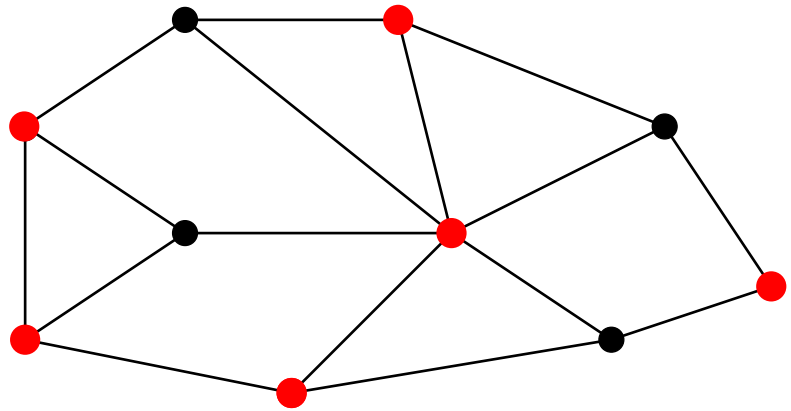
On kernel size (1): Vertex Cover

- Can we remove from G at most k edges to obtain an edge-free graph?



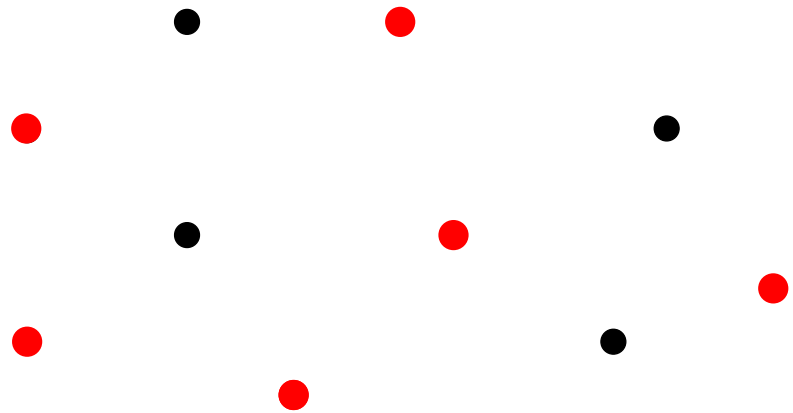
On kernel size (1): Vertex Cover

- Can we remove from G at most k edges to obtain an edge-free graph?



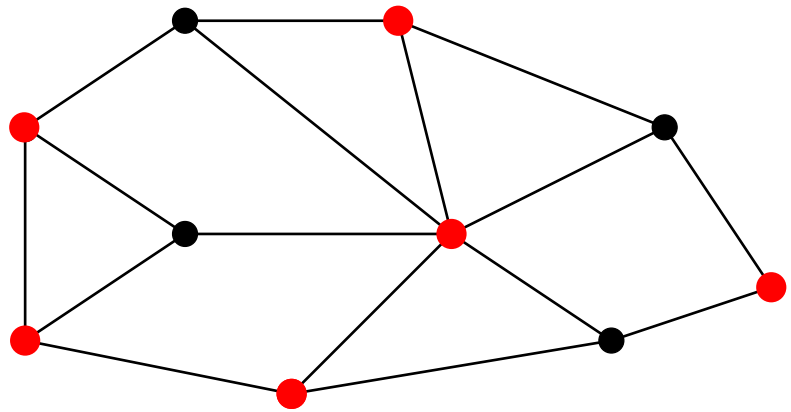
On kernel size (1): Vertex Cover

- Can we remove from G at most k edges to obtain an edge-free graph?



On kernel size (1): Vertex Cover

- Can we remove from G at most k edges to obtain an edge-free graph?

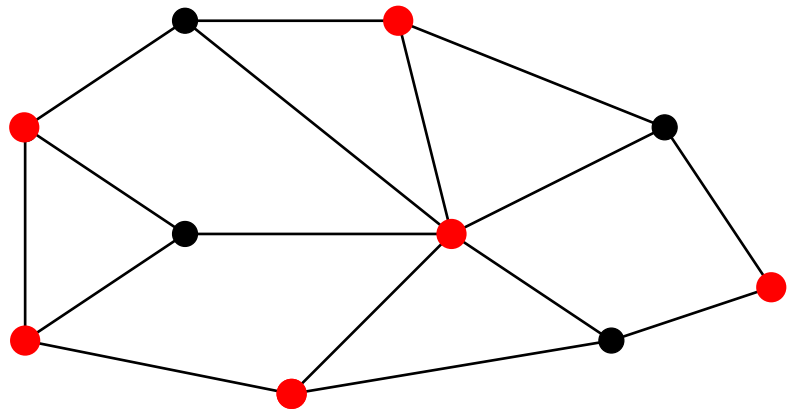


Theorem

- Vertex Cover parameterized by solution size has a $O(k^2)$ vertex kernel [Buss]

On kernel size (1): Vertex Cover

- Can we remove from G at most k edges to obtain an edge-free graph?

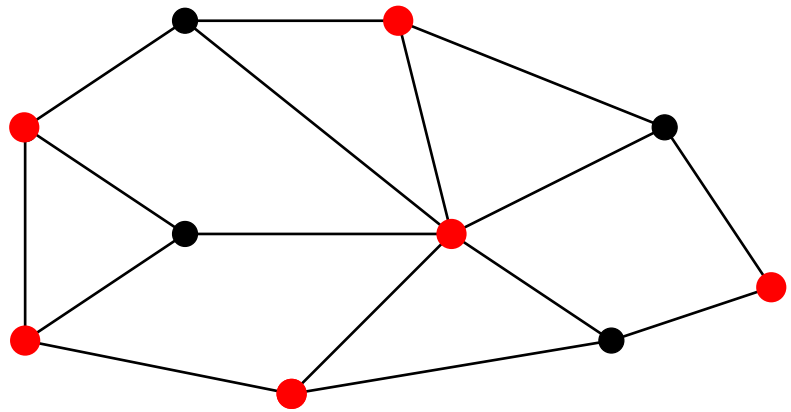


Theorem

- Vertex Cover parameterized by solution size has a $O(k^2)$ vertex kernel [Buss]
- Vertex Cover parameterized by solution size has a $2k$ vertex kernel e.g. via linear programming

On kernel size (1): Vertex Cover

- Can we remove from G at most k edges to obtain an edge-free graph?



Theorem

- Vertex Cover parameterized by solution size has a $O(k^2)$ vertex kernel [Buss]
- Vertex Cover parameterized by solution size has a $2k$ vertex kernel e.g. via linear programming
- Vertex Cover parameterized by solution size has no $O(k^{2-\epsilon})$ edge kernel unless $coNP \subseteq NP/poly$ [Dell, van Melkebeek]

On kernel size (2)

Theorem [Folkore]

A parameterized problem has a kernelization algorithm iff it can be solved in time $f(k).n^{O(1)}$ where k is the parameter and n the data-size (a.k.a the problem is Fixed Parameter Tractable).

On kernel size (2)

Theorem [Folkore]

A parameterized problem has a kernelization algorithm iff it can be solved in time $f(k).n^{O(1)}$ where k is the parameter and n the data-size (a.k.a the problem is Fixed Parameter Tractable).

But not every parameterized problem is FPT

- Clique parameterized by the solution size is $W[1]$-complete
- Dominating Set parameterized by the solution size is $W[2]$-complete

$$FPT \subseteq W[1] \subseteq W[2] \cdots \subseteq W[P] \subseteq XP$$

On kernel size (2)

Theorem [Folkore]

A parameterized problem has a kernelization algorithm iff it can be solved in time $f(k).n^{O(1)}$ where k is the parameter and n the data-size (a.k.a the problem is Fixed Parameter Tractable).

But not every parameterized problem is FPT

- Clique parameterized by the solution size is $W[1]$-complete
- Dominating Set parameterized by the solution size is $W[2]$-complete

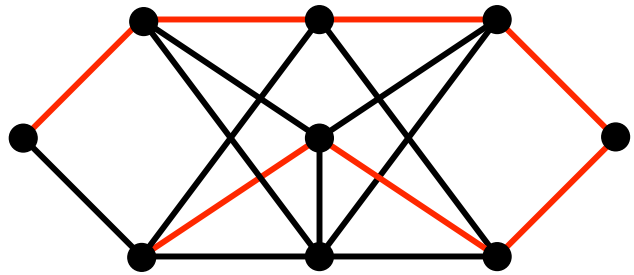
$$FPT \subseteq W[1] \subseteq W[2] \cdots \subseteq W[P] \subseteq XP$$

Corollary

Clique and Dominating Set parameterized by the solution size does not have a kernelization algorithm.

On kernel size (3): exponential lower bound

- A graph $G = (V, E)$ and a parameter $k \in \mathbb{N}$
- Does G contains a path of length k?

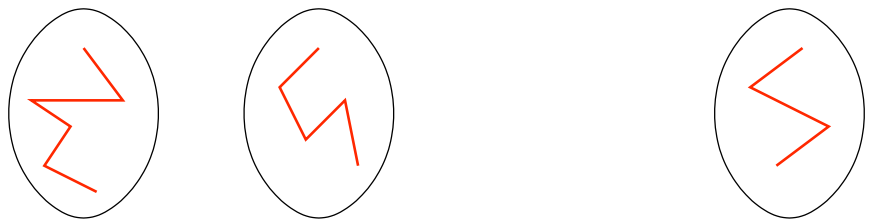


Longest Path is NP-Complete (cf. Hamiltonian Path) but can be solved in FPT time.

On kernel size (3): exponential lower bound

Hypothesis There exists a kernelization algorithm $\mathcal{A}$ for Longest Path which computes a polynomial size kernel.

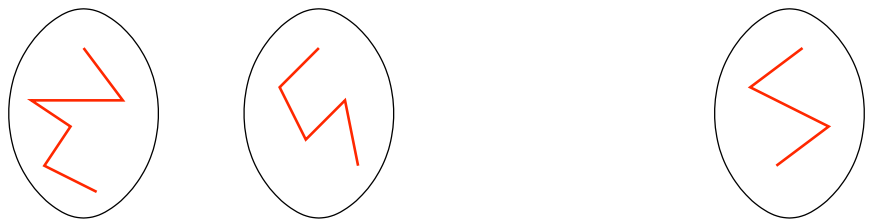
- Let (G, k) be $(G_1, k) \oplus (G_2, k) \oplus \ \ldots \ \oplus (G_t, k)$



On kernel size (3): exponential lower bound

Hypothesis There exists a kernelization algorithm $\mathcal{A}$ for Longest Path which computes a polynomial size kernel.

- Let (G, k) be $(G_1, k) \oplus (G_2, k) \oplus \ \ldots \ \oplus (G_t, k)$



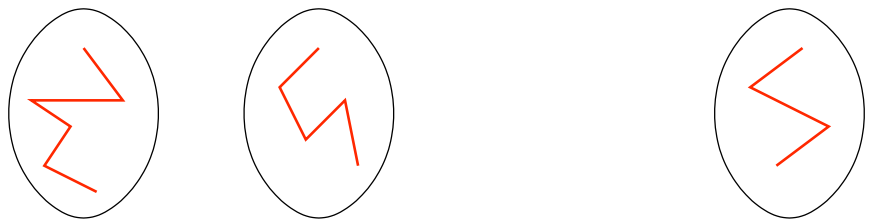
Observation : (G, k) has a path of length k iff $\exists i$ st G_i has a path of length k.

Question : Can it be the case that $\mathcal{A}$ identifies in polynomial time which G_i has a path of length k ?

On kernel size (3): exponential lower bound

Hypothesis There exists a kernelization algorithm $\mathcal{A}$ for Longest Path which computes a polynomial size kernel.

- Let (G, k) be $(G_1, k) \oplus (G_2, k) \oplus \ \ldots \ \oplus (G_t, k)$



Observation : (G, k) has a path of length k iff $\exists i$ st G_i has a path of length k.

Question : Can it be the case that $\mathcal{A}$ identifies in polynomial time which G_i has a path of length k ?

Theorem [Fortnow, Santhaman + Bodlaender, Downey, Fellows, Hermelin] If a parameter problem is OR-composable and has a polynomial size kernel, then $PH = \Sigma_p^3$

On kernel size (4)

For a few years now a lower bound machinery is been developped:

- OR-composition
- Parameter preserving and polynomial transformation
- cross-composition
- weak-distillation
- ...

Meanwhile, recent results propose meta-theorem to prove polynomial upper-bounds

On parameter value

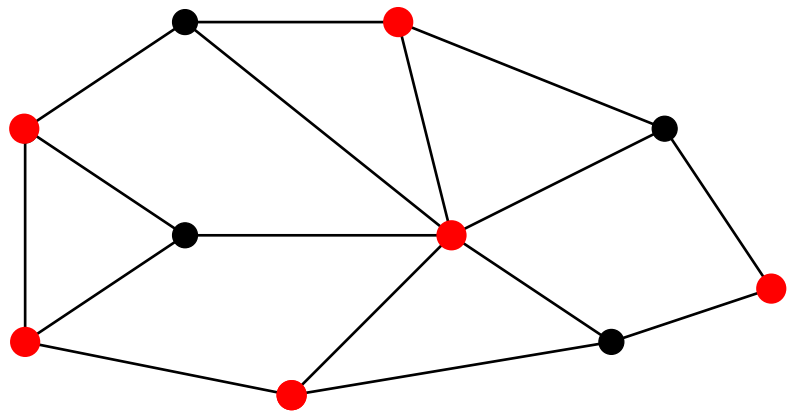
Lemma Let k_1 and k_2 be two parameters such $k_1 \leqslant k_2$.
If a problem Π parameterized by k_2 has no polynomial kernel, then it has no polynomial kernel when parameterized by k_1

On parameter value

Lemma Let k_1 and k_2 be two parameters such $k_1 \leqslant k_2$.
If a problem Π parameterized by k_2 has no polynomial kernel,
then it has no polynomial kernel when parameterized by k_1

$\Rightarrow$ Hierarchies of parameters:

Vertex Cover $\geqslant$ Feedback Vertex set $\geqslant$ Treewidth

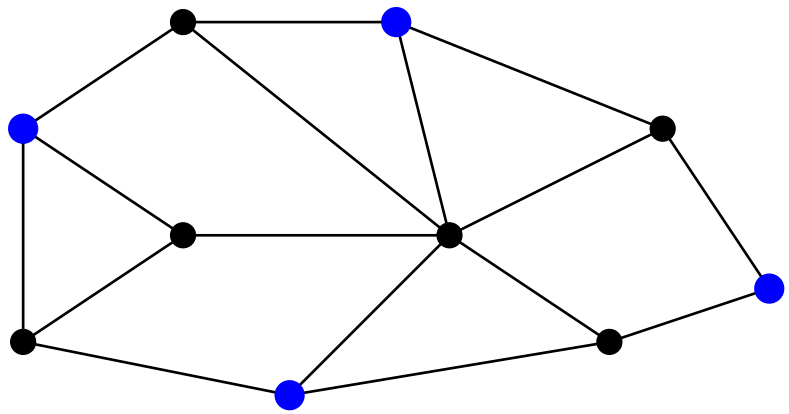


On parameter value

Lemma Let k_1 and k_2 be two parameters such $k_1 \leqslant k_2$.
If a problem Π parameterized by k_2 has no polynomial kernel,
then it has no polynomial kernel when parameterized by k_1

$\Rightarrow$ Hierarchies of parameters:

VERTEX COVER $\geqslant$ FEEDBACK VERTEX SET $\geqslant$ TREEWIDTH

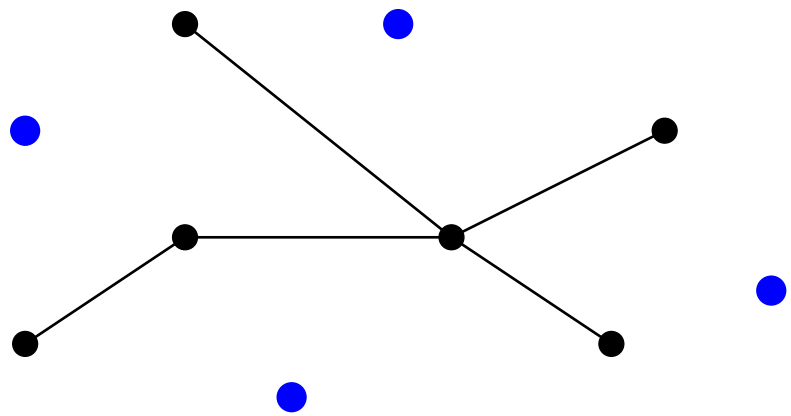


On parameter value

Lemma Let k_1 and k_2 be two parameters such $k_1 \leqslant k_2$.
If a problem Π parameterized by k_2 has no polynomial kernel,
then it has no polynomial kernel when parameterized by k_1

$\Rightarrow$ Hierarchies of parameters:

Vertex Cover $\geqslant$ Feedback Vertex set $\geqslant$ Treewidth



On parameter value

Lemma Let k_1 and k_2 be two parameters such $k_1 \leqslant k_2$.
If a problem Π parameterized by k_2 has no polynomial kernel,
then it has no polynomial kernel when parameterized by k_1

$\Rightarrow$ Hierarchies of parameters:

$$\textsc{Vertex Cover} \geqslant \textsc{Feedback Vertex set} \geqslant \textsc{Treewidth}$$

Theorem Vertex Cover parameterized by

- Vertex Cover has a linear kernel
- Feedback Vertex Set has a cubic kernel
- Treewidth has no polynomial kernel

where the transition does take place ?

Complexity time of kernelization

How fast an optimal kernel can be computed?

- most effort has be put on optimizing kernel size, and only a few on computation time
- [Niedermeier et al.] have recently shown that a non-exhaustive application of the reduction rules may lead to a faster computation without altering too much the kernel size.

Which guarantee to preserve?

$\Rightarrow$ Approximative kernelization (α-fidely kernel, $\alpha \leqslant 1$)

1. transform in polytime an parameterized instance (x, k) into an instance (x', k') such that
   - $k' \leqslant k$
   - $|x'| \leqslant g(k, \alpha)$

Which guarantee to preserve?

$\Rightarrow$ Approximative kernelization (α-fidely kernel, $\alpha \leqslant 1$)

1. transform in polytime an parameterized instance (x, k) into an instance (x', k') such that
   - $k' \leqslant k$
   - $|x'| \leqslant g(k, \alpha)$
2. $(x, k/\alpha)$ is a YES-instance $\Rightarrow$ (x', k') is a YES-instance

Which guarantee to preserve?

$\Rightarrow$ Approximative kernelization (α-fidely kernel, $\alpha \leqslant 1$)

1. transform in polytime an parameterized instance (x, k) into an instance (x', k') such that
   - $k' \leqslant k$
   - $|x'| \leqslant g(k, \alpha)$
2. $(x, k/\alpha)$ is a YES-instance $\Rightarrow$ (x', k') is a YES-instance
3. (x', k') is a YES-instance $\Rightarrow$ (x, k) is a YES-instance

Which guarantee to preserve?

$\Rightarrow$ Approximative kernelization (α-fidely kernel, $\alpha \leqslant 1$)

1. transform in polytime an parameterized instance (x, k) into an instance (x', k') such that
   - $k' \leqslant k$
   - $|x'| \leqslant g(k, \alpha)$
2. $(x, k/\alpha)$ is a YES-instance $\Rightarrow$ (x', k') is a YES-instance
3. (x', k') is a YES-instance $\Rightarrow$ (x, k) is a YES-instance

Observation for $\alpha = 1$, this is the classical kernel

Which guarantee to preserve?

$\Rightarrow$ Approximative kernelization (α-fidely kernel, $\alpha \leqslant 1$)

1. transform in polytime an parameterized instance (x, k) into an instance (x', k') such that
   - $k' \leqslant k$
   - $|x'| \leqslant g(k, \alpha)$
2. $(x, k/\alpha)$ is a YES-instance $\Rightarrow$ (x', k') is a YES-instance
3. (x', k') is a YES-instance $\Rightarrow$ (x, k) is a YES-instance

Observation for $\alpha = 1$, this is the classical kernel

Theorem [Fellows, Shachnai]
Vertex Cover has a $2k(2-\alpha)/\alpha$-fidelity kernelization

Experimental studies

- Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments by F. Abu-Khzam, R. Collins, M. Fellows, M. Langston, W. Suters and C. Symons, 2004.

  > *Vertex Cover has many real-world applications, including many in the field of bioinformatics. It can be used in the construction of phylogenetic trees, in phenotype identification, and in analysis of microarray data, just to name a few.*
  > *[. . .] One of the applications to which we have applied our codes is the problem of finding phylogenetic trees based on protein domains. The graphs that we utilized were obtained based on data from NCBI and SWISS-PROT, well known open-source repositories of biological data.*

Experimental studies

- Kernelization Algorithms for the Vertex Cover Problem: Theory and Experiments by F. Abu-Khzam, R. Collins, M. Fellows, M. Langston, W. Suters and C. Symons, 2004.

  > *Vertex Cover has many real-world applications, including many in the field of bioinformatics. It can be used in the construction of phylogenetic trees, in phenotype identification, and in analysis of microarray data, just to name a few.*
  > *[. . .] One of the applications to which we have applied our codes is the problem of finding phylogenetic trees based on protein domains. The graphs that we utilized were obtained based on data from NCBI and SWISS-PROT, well known open-source repositories of biological data.*

- Partial Kernelization for Rank Aggregation: Theory and Experiments by N. Betzler, R. Bredereck and R. Niedermeier, 2010.

Conclusion

- Kernelization is an attempt to provide a rigorous mathematical framework to explain the power of heuristics.
- Fast and important development in the last 5 years
- It is now spreading in many different areas ranging from theoretical computer science to applied domains

Conclusion

- Kernelization is an attempt to provide a rigorous mathematical framework to explain the power of heuristics.
- Fast and important development in the last 5 years
- It is now spreading in many different areas ranging from theoretical computer science to applied domains

- importance of the choice of the parameter
- interleaving kernelization and other techniques yields really efficient algorithms
- exhaustive application or not of the reduction rules ?

Conclusion

- Kernelization is an attempt to provide a rigorous mathematical framework to explain the power of heuristics.
- Fast and important development in the last 5 years
- It is now spreading in many different areas ranging from theoretical computer science to applied domains

- importance of the choice of the parameter
- interleaving kernelization and other techniques yields really efficient algorithms
- exhaustive application or not of the reduction rules ?

- May be interesting to confront heuristics developed in the context of NUMEV with kernelization techniques

Conclusion

- Kernelization is an attempt to provide a rigorous mathematical framework to explain the power of heuristics.
- Fast and important development in the last 5 years
- It is now spreading in many different areas ranging from theoretical computer science to applied domains

- importance of the choice of the parameter
- interleaving kernelization and other techniques yields really efficient algorithms
- exhaustive application or not of the reduction rules ?

- May be interesting to confront heuristics developed in the context of NUMEV with kernelization techniques

- ANR Project AGAPE (2009-2013)
- KERNEL Project (Chercheur d'Avenir - Languedoc-Roussillon - 2012-2015)