Exploiting Graph Stucture in Multivariate Algorithmics

Rolf Niedermeier

TU Berlin
Institut für Softwaretechnik und Theoretische Informatik
Algorithmics and Complexity Theory
http://www.akt.tu-berlin.de

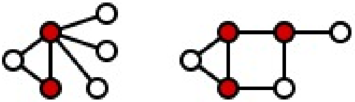
Durham University, Graph Theory and Interactions, July 2013

Motivating (Standard) Example I

Established efficiency race for NP-hard **Vertex Cover** problem:

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.

Task: Find a subset of vertices $C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in C.

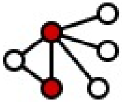


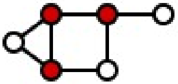
Motivating (Standard) Example I

Established efficiency race for NP-hard **Vertex Cover** problem:

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.

Task: Find a subset of vertices $C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in C.





Currently best upper bound: $O(1.274^k + k|V|)$ time.

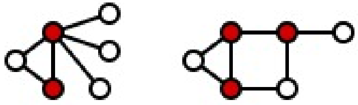
[Chen, Kanj, Xia: Theor. Comput. Sci. 2010]

Motivating (Standard) Example I

Established efficiency race for NP-hard **Vertex Cover** problem:

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.

Task: Find a subset of vertices $C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in C.



Currently best upper bound: $O(1.274^k + k|V|)$ time.

[Chen, Kanj, Xia: Theor. Comput. Sci. 2010]

Grain of salt: In many applications, the parameter k is not small and grows with the graph size.

Motivating Example Vertex Cover

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.

Task: Find a subset of vertices $C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in C.

Now: *New (above guarantee) parameter* for Vertex Cover: $k' := k - LP$, where LP denotes the value of the linear programming relaxation of the standard ILP for Vertex Cover...

[Narayanaswamy et al., STACS 2012; updated version Lokshtanov et al., arXiv 2012]

Motivating Example Vertex Cover

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.

Task: Find a subset of vertices $C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in C.

Now: *New (above guarantee) parameter* for Vertex Cover: $k' := k - LP$, where LP denotes the value of the linear programming relaxation of the standard ILP for Vertex Cover...

[Narayanaswamy et al., STACS 2012; updated version Lokshtanov et al., arXiv 2012]

So, LP bound is a guaranteed lower bound for solution size!

Clearly, k' is "stronger" than k.

Motivating Example Vertex Cover

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.

Task: Find a subset of vertices $C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in C.

Now: *New (above guarantee) parameter* for Vertex Cover: $k' := k - LP$, where LP denotes the value of the linear programming relaxation of the standard ILP for Vertex Cover...

[Narayanaswamy et al., STACS 2012; updated version Lokshtanov et al., arXiv 2012]

So, LP bound is a guaranteed lower bound for solution size!

Clearly, k' is “stronger” than k.

Central result: Vertex Cover solvable in $2.32^{k'} \cdot (|V| + |E|)^{O(1)}$ time.

Motivating Example Vertex Cover

Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.

Task: Find a subset of vertices $C \subseteq V$ with k or fewer vertices such that each edge in E has at least one of its endpoints in C.

Now: *New (above guarantee) parameter* for Vertex Cover: $k' := k - LP$, where LP denotes the value of the linear programming relaxation of the standard ILP for Vertex Cover...

[Narayanaswamy et al., STACS 2012; updated version Lokshtanov et al., arXiv 2012]

So, LP bound is a guaranteed lower bound for solution size!

Clearly, k' is "stronger" than k.

Central result: Vertex Cover solvable in $2.32^{k'} \cdot (|V| + |E|)^{O(1)}$ time.

$\rightsquigarrow$ **Our general theme:** Are there structures (that is, parameterizations) that can be exploited for deriving "efficient" solutions for NP-hard problems?

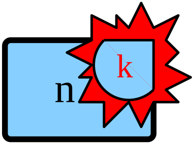
Parameterized Algorithmics in a Nutshell

NP-hard problem X: Input size n and problem parameter k.

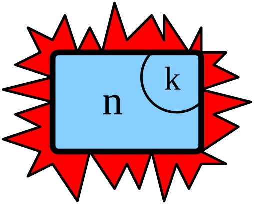
If there is an algorithm solving X in time

$$f(k) \cdot n^{O(1)},$$

then X is called **fixed-parameter tractable (FPT)**:



instead of



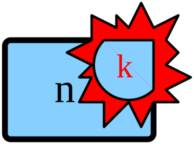
Parameterized Algorithmics in a Nutshell

NP-hard problem X: Input size n and problem parameter k.

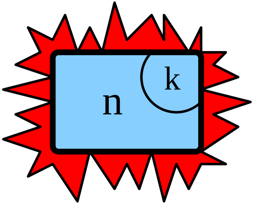
If there is an algorithm solving X in time

$$f(k) \cdot n^{O(1)},$$

then X is called **fixed-parameter tractable (FPT)**:



instead of



Completeness program developed by Downey and Fellows (1999).

$$\text{FPT} \subseteq \overbrace{\text{W[1]} \subseteq \text{W[2]} \subseteq \ldots \subseteq \text{W[P]} \subseteq \text{XP}}^{\text{Presumably fixed-parameter intractable}}$$

Parameterized Complexity Hierarchy

FPT vs W[1]-hard vs para-NP-hard:

- Vertex Cover parameterized by solution size is FPT;
- Clique parameterized by solution size is W[1]-hard (but in XP);
- k-Coloring is para-NP-hard (and thus not in XP unless P=NP) (because it is NP-hard for $k = 3$ colors (that is, constant parameter value)).

Parameterized Complexity Hierarchy

FPT vs W[1]-hard vs para-NP-hard:

- Vertex Cover parameterized by solution size is FPT;
- Clique parameterized by solution size is W[1]-hard (but in XP);
- k-Coloring is para-NP-hard (and thus not in XP unless P=NP) (because it is NP-hard for $k = 3$ colors (that is, constant parameter value)).

"Function battle" concerning allowed running time:

FPT: $f(k) \cdot n^{O(1)}$ **vs** **XP:** $f(k) \cdot n^{g(k)}$

Parameterized Complexity Hierarchy

FPT vs W[1]-hard vs para-NP-hard:

- Vertex Cover parameterized by solution size is FPT;
- Clique parameterized by solution size is W[1]-hard (but in XP);
- k-Coloring is para-NP-hard (and thus not in XP unless P=NP) (because it is NP-hard for $k = 3$ colors (that is, constant parameter value)).

"Function battle" concerning allowed running time:

$$\textbf{FPT: } f(k) \cdot n^{O(1)} \quad \textbf{vs} \quad \textbf{XP: } f(k) \cdot n^{g(k)}$$

Assumption: FPT $\neq$ W[1]

For instance, if W[1]=FPT then 3-SAT for a Boolean formula F with n variables can be solved in $2^{o(n)} \cdot |F|^{O(1)}$ time.

The “Art” of Parameter Identification

Central question: How to find relevant parameterizations?

Central fact: One problem may have a large number of different (relevant) parameterizations, that is, structures to exploit...

The “Art” of Parameter Identification

Central question: How to find relevant parameterizations?

Central fact: One problem may have a large number of different (relevant) parameterizations, that is, structures to exploit...

Consequence: We achieve a more fine-grained but also more complicated picture of the computational complexity of problems.

$\leadsto$ Parameterized algorithmics goes multivariate...

The "Art" of Parameter Identification

Central question: How to find relevant parameterizations?

Central fact: One problem may have a large number of different (relevant) parameterizations, that is, structures to exploit...

Consequence: We achieve a more fine-grained but also more complicated picture of the computational complexity of problems.

$\rightsquigarrow$ Parameterized algorithmics goes multivariate...

Note: Parameterizations typically are of "structural nature", modelling (sometimes hidden) properties that input instances may have...

The “Art” of Parameter Identification

Central question: How to find relevant parameterizations?

Central fact: One problem may have a large number of different (relevant) parameterizations, that is, structures to exploit...

Consequence: We achieve a more fine-grained but also more complicated picture of the computational complexity of problems.

$\rightsquigarrow$ Parameterized algorithmics goes multivariate...

Note: Parameterizations typically are of “structural nature”, modelling (sometimes hidden) properties that input instances may have...

Basic philosophy: Different parameterizations allow for different views, resulting in a “holistic” approach to complexity analysis.

Revisiting hardness proofs, **deconstruct intractability**!

A Theoretical Way of Spotting Parameters

Call parameter k_1 **stronger than** parameter k_2 if there is a constant c such that $k_1 \leq c \cdot k_2$ for all inputs, and there is no constant d such that $k_2 \leq d \cdot k_1$ for all inputs.
(Analogously: k_2 is **weaker than** k_1).

A Theoretical Way of Spotting Parameters

Call parameter k_1 **stronger than** parameter k_2 if there is a constant c such that $k_1 \leq c \cdot k_2$ for all inputs, and there is no constant d such that $k_2 \leq d \cdot k_1$ for all inputs.
(Analogously: k_2 is **weaker than** k_1).

Examples:

- Average vertex degree of a graph is stronger than maximum vertex degree.

A Theoretical Way of Spotting Parameters

Call parameter k_1 **stronger than** parameter k_2 if there is a constant c such that $k_1 \leq c \cdot k_2$ for all inputs, and there is no constant d such that $k_2 \leq d \cdot k_1$ for all inputs.
(Analogously: k_2 is **weaker than** k_1).

Examples:

- Average vertex degree of a graph is stronger than maximum vertex degree.
- Treewidth is a stronger parameter than vertex cover number of a graph.

A Theoretical Way of Spotting Parameters

Call parameter k_1 **stronger than** parameter k_2 if there is a constant c such that $k_1 \leq c \cdot k_2$ for all inputs, and there is no constant d such that $k_2 \leq d \cdot k_1$ for all inputs.
(Analogously: k_2 is **weaker than** k_1).

Examples:

- Average vertex degree of a graph is stronger than maximum vertex degree.
- Treewidth is a stronger parameter than vertex cover number of a graph.
- Also: Single parameter k_1 is stronger than combined parameter $k_1 + k_2$ whatever k_2 is.

Goals for Stronger and Weaker Parameterizations

Primary goals:

1. Whenever a problem is fixed-parameter tractable with respect to a parameter k_1, then try to also show fixed-parameter tractability for a **stronger** parameter k_2.

Goals for Stronger and Weaker Parameterizations

Primary goals:

1. Whenever a problem is fixed-parameter tractable with respect to a parameter k_1, then try to also show fixed-parameter tractability for a **stronger** parameter k_2.
2. If a problem is W[1]-hard with respect to a parameter k_1, then try to show fixed-parameter tractability for a **weaker** parameter k_2.

Goals for Stronger and Weaker Parameterizations

Primary goals:

1. Whenever a problem is fixed-parameter tractable with respect to a parameter k_1, then try to also show fixed-parameter tractability for a **stronger** parameter k_2.
2. If a problem is W[1]-hard with respect to a parameter k_1, then try to show fixed-parameter tractability for a **weaker** parameter k_2.

Secondary goals:

- Can similar upper bounds be achieved for stronger parameters?
- Provide a “complete” map of a problem’s (parameterized) computational complexity with respect to various parameterizations (partially) ordered by their respective “strength”.

$\rightsquigarrow$ **Profiling** of NP-hard (graph) problems...

First Example: Finding 2-Clubs

NP-hard ***s*-Club** problem (occurring in the analysis of social and biological networks):

Input A graph $G = (V, E)$ and an integer k.

Question Is there a vertex set $V' \subseteq V$ of size at least k such that $G[V']$ has diameter at most s?

- 1-Club is equivalent to Clique.

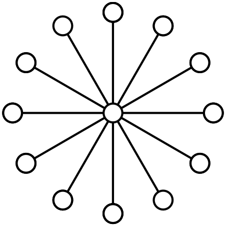
First Example: Finding 2-Clubs

NP-hard ***s*-Club** problem (occurring in the analysis of social and biological networks):

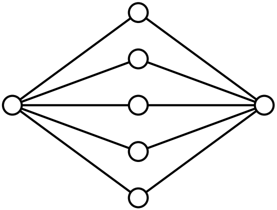
Input A graph $G = (V, E)$ and an integer k.

Question Is there a vertex set $V' \subseteq V$ of size at least k such that $G[V']$ has diameter at most s?

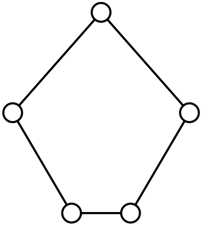
- 1-Club is equivalent to Clique.
- We focus on 2-Club:



star



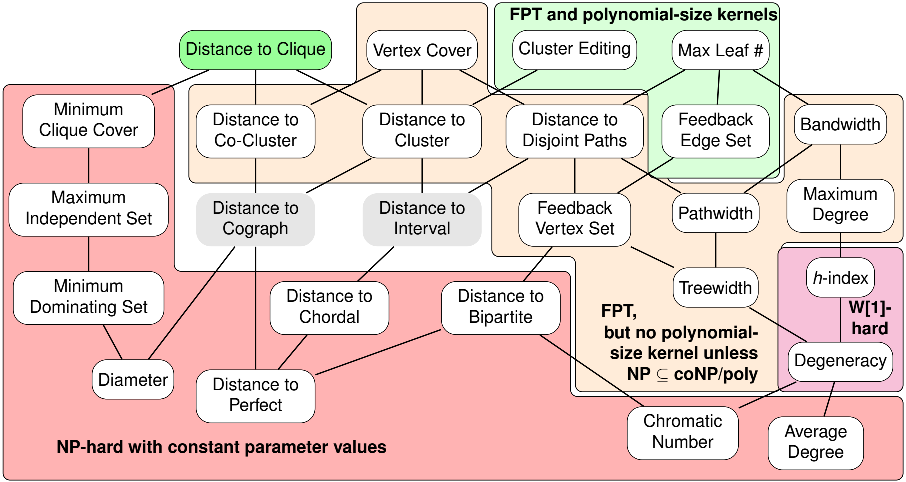
diamond



C_5

Navigating Through Parameter Space: 2-Clubs

[Hartung, Komusiewicz, Nichterlein, IPEC 2012 + SOFSEM 2013].

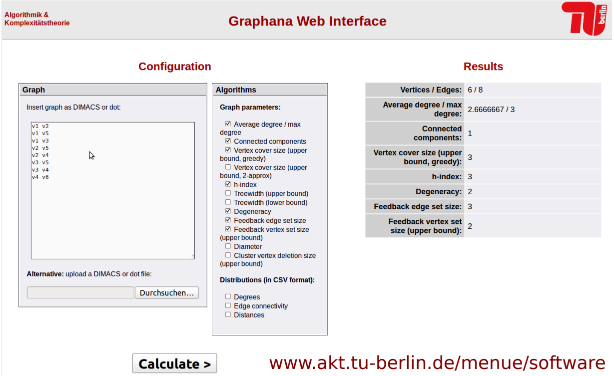


A More Pragmatic Way of Spotting Parameters

A simple way to spot interesting parameterizations (structure) in real-world graph problems: Measure “all” possible parameters...:

A More Pragmatic Way of Spotting Parameters

A simple way to spot interesting parameterizations (structure) in real-world graph problems: Measure “all” possible parameters...:
Use tool Graphana for data-driven parameterization:



www.akt.tu-berlin.de/menue/software

Outline of the Remaining Talk

We discuss three recent examples for graph problems where structure detection and parameterized complexity analysis were instrumental:

- Arising from biological network analysis: **Highly Connected Deletion** problem;
- Arising from social network analysis: **Graph Anonymization** problem.
- Arising from incremental clustering: **Incremental Conservative** k**-List Coloring** problem

Case Study: Highly Connected Deletion

Typical graph clustering situation:
Task: Partition a graph into clusters such that

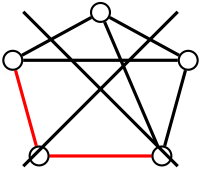
- each cluster is **dense** and
- there are few edges between clusters.

Case Study: Highly Connected Deletion

Typical graph clustering situation:
Task: Partition a graph into clusters such that

- each cluster is **dense** and
- there are few edges between clusters.

$\rightsquigarrow$ **Definition:** [Hartuv & Shamir, Inf. Proc. Letters '00]
A graph with n vertices is **highly connected** if more than $n/2$ edges need to be deleted to make it disconnected.

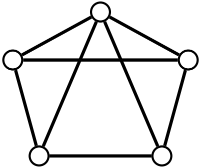


Case Study: Highly Connected Deletion

Typical graph clustering situation:
Task: Partition a graph into clusters such that

- each cluster is **dense** and
- there are few edges between clusters.

⇝ **Definition:** [Hartuv & Shamir, Inf. Proc. Letters '00]
A graph with n vertices is **highly connected** if more than $n/2$ edges need to be deleted to make it disconnected.

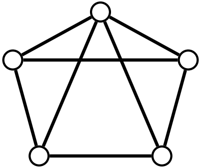


Case Study: Highly Connected Deletion

Typical graph clustering situation:
Task: Partition a graph into clusters such that

- each cluster is **dense** and
- there are few edges between clusters.

$\rightsquigarrow$ **Definition:** [Hartuv & Shamir, Inf. Proc. Letters '00]
A graph with n vertices is **highly connected** if more than $n/2$ edges need to be deleted to make it disconnected.



Properties:

- diameter two;
- each vertex has degree $\geq \lfloor n/2 \rfloor$.

A MinCut Heuristic for Highly-Connected Clustering

Task: Partition the network into clusters such that

- each cluster is highly connected and
- there are few edges between clusters.

Challenge: How to find highly connected clusters?

A MinCut Heuristic for Highly-Connected Clustering

Task: Partition the network into clusters such that

- each cluster is highly connected and
- there are few edges between clusters.

Challenge: How to find highly connected clusters?

Min-Cut Algorithm: [Hartuv & Shamir, IPL 2000]

Input: $G = (V, E)$

(A, B)= min-cut(G)
if (A, B) has $> |V|/2$ edges **: output** V
else: recurse on $G[A]$ and $G[B]$

A MinCut Heuristic for Highly-Connected Clustering

Task: Partition the network into clusters such that

- each cluster is highly connected and
- there are few edges between clusters.

Challenge: How to find highly connected clusters?

Min-Cut Algorithm: [Hartuv & Shamir, IPL 2000]

Input: $G = (V, E)$

(A, B)= min-cut(G)
if (A, B) has $> |V|/2$ edges **: output** V
else: recurse on $G[A]$ and $G[B]$

Biological applications:

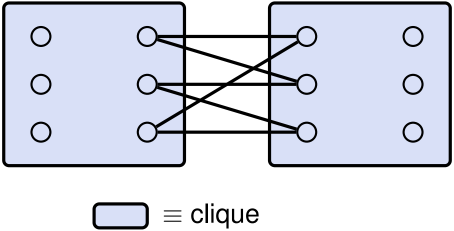
- Clustering cDNA fingerprints
- Complex identification in protein–interaction networks
- Hierarchical clustering of protein sequences
- Clustering regulatory RNA structures

Edge Coverage

Task: Partition the network into clusters such that

- each cluster is highly connected and
- the number of **edges between clusters** is minimal.

Does the MinCut heuristic achieve **the second goal**?
$\rightsquigarrow$ Comparison with optimal solution...



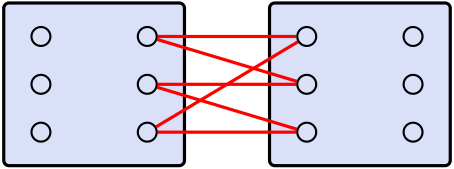
$\equiv$ clique

Edge Coverage

Task: Partition the network into clusters such that

- each cluster is highly connected and
- the number of **edges between clusters** is minimal.

Does the MinCut heuristic achieve **the second goal**?
$\rightsquigarrow$ Comparison with optimal solution...



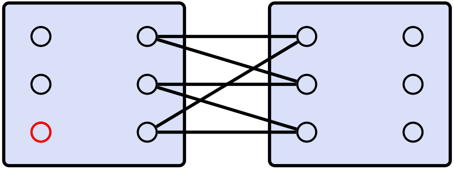

$\equiv$ clique

Edge Coverage

Task: Partition the network into clusters such that

- each cluster is highly connected and
- the number of **edges between clusters** is minimal.

Does the MinCut heuristic achieve **the second goal**?
$\rightsquigarrow$ Comparison with optimal solution...




$\equiv$ clique

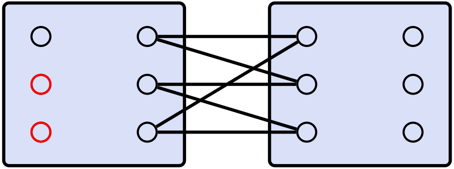
Edge Coverage

Task: Partition the network into clusters such that

- each cluster is highly connected and
- the number of **edges between clusters** is minimal.

Does the MinCut heuristic achieve **the second goal**?

$\rightsquigarrow$ Comparison with optimal solution...



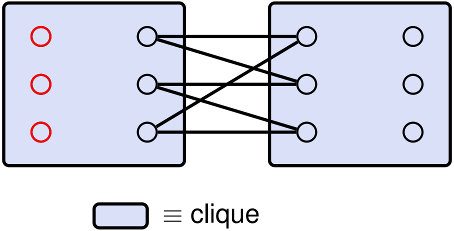

$\equiv$ clique

Edge Coverage

Task: Partition the network into clusters such that

- each cluster is highly connected and
- the number of **edges between clusters** is minimal.

Does the MinCut heuristic achieve **the second goal**?
$\rightsquigarrow$ Comparison with optimal solution...



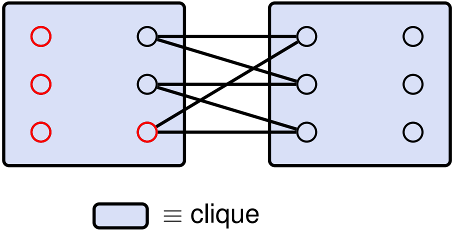
$\equiv$ clique

Edge Coverage

Task: Partition the network into clusters such that

- each cluster is highly connected and
- the number of **edges between clusters** is minimal.

Does the MinCut heuristic achieve **the second goal**?
$\rightsquigarrow$ Comparison with optimal solution...

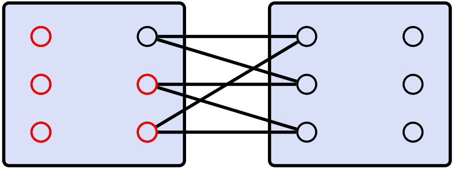


Edge Coverage

Task: Partition the network into clusters such that

- each cluster is highly connected and
- the number of **edges between clusters** is minimal.

Does the MinCut heuristic achieve **the second goal**?
$\rightsquigarrow$ Comparison with optimal solution...

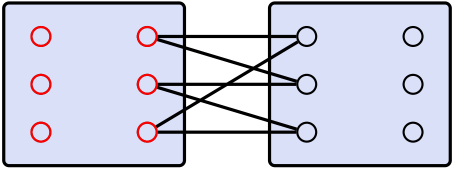


Edge Coverage

Task: Partition the network into clusters such that

- each cluster is highly connected and
- the number of **edges between clusters** is minimal.

Does the MinCut heuristic achieve **the second goal**?
$\rightsquigarrow$ Comparison with optimal solution...



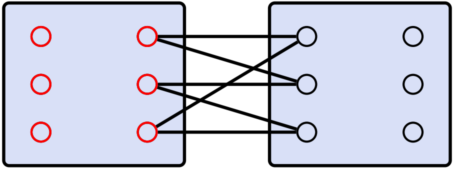

$\equiv$ clique

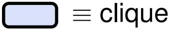
Edge Coverage

Task: Partition the network into clusters such that

- each cluster is highly connected and
- the number of **edges between clusters** is minimal.

Does the MinCut heuristic achieve **the second goal**?
⇝ Comparison with optimal solution...





⇝ MinCut heuristic may delete $\Theta(\mathrm{OPT}^2)$ many edges!
⇝ New goal: find optimal clustering

Complexity of Highly Connected Deletion

Highly Connected Deletion
Input: An undirected graph.
Task: Delete a minimum number of edges such that each remaining connected component is highly connected.

Theorem: Highly Connected Deletion is NP-hard even on 4-regular graphs.

Complexity of Highly Connected Deletion

Highly Connected Deletion
Input: An undirected graph.
Task: Delete a minimum number of edges such that each remaining connected component is highly connected.

Theorem: Highly Connected Deletion is NP-hard even on 4-regular graphs.

Theorem: If the Exponential Time Hypothesis (ETH) is true, then **Highly Connected Deletion** cannot be solved within $2^{o(m)}\,\mathrm{poly}(n)$ time or $2^{o(n)}\,\mathrm{poly}(n)$ time.

$m :=$ number of edges
$n :=$ number of vertices

Data Reduction Rules

Idea 1: Find vertex sets that are **inseparable** because any cut of this set has size $> k$
$\rightsquigarrow$ **Too-Connected-Rule:** If G contains an inseparable vertex set S of size at least $2k$, then do the following. If $G[S]$ is not highly connected, return “no”. Otherwise, remove S from G and adapt k correspondingly.

Data Reduction Rules

Idea 1: Find vertex sets that are **inseparable** because any cut of this set has size $> k$
$\rightsquigarrow$ **Too-Connected-Rule:** If G contains an inseparable vertex set S of size at least $2k$, then do the following. If $G[S]$ is not highly connected, return "no". Otherwise, remove S from G and adapt k correspondingly.

Idea 2: Find highly connected clusters that are large compared to the number of "outgoing" edges
$\rightsquigarrow$ $D(S) :=$ edge cut between S and $V \setminus S$

Data Reduction Rules

Idea 1: Find vertex sets that are **inseparable** because any cut of this set has size $> k$
$\rightsquigarrow$ **Too-Connected-Rule:** If G contains an inseparable vertex set S of size at least $2k$, then do the following. If $G[S]$ is not highly connected, return "no". Otherwise, remove S from G and adapt k correspondingly.

Idea 2: Find highly connected clusters that are large compared to the number of "outgoing" edges
$\rightsquigarrow$ $D(S) :=$ edge cut between S and $V \setminus S$

Small-Cut-Rule: If G contains a vertex set S such that

- $|S| \geq 4$,
- $G[S]$ is highly connected, and
- $|D(S)| \leq 0.3 \cdot \sqrt{|S|}$,

then remove S from G.

Data Reduction Rules

Idea 1: Find vertex sets that are **inseparable** because any cut of this set has size $> k$
$\rightsquigarrow$ **Too-Connected-Rule:** If G contains an inseparable vertex set S of size at least $2k$, then do the following. If $G[S]$ is not highly connected, return "no". Otherwise, remove S from G and adapt k correspondingly.

Idea 2: Find highly connected clusters that are large compared to the number of "outgoing" edges
$\rightsquigarrow$ $D(S) :=$ edge cut between S and $V \setminus S$

Small-Cut-Rule: If G contains a vertex set S such that

- $|S| \geq 4$,
- $G[S]$ is highly connected, and
- $|D(S)| \leq 0.3 \cdot \sqrt{|S|}$,

then remove S from G.

Theorem: Highly Connected Deletion admits polynomial-time data reduction to an equivalent instance with $\leq 10 \cdot k^{1.5}$ vertices.

FPT Algorithm and Further Data Reduction

Combination of branching, data reduction, and dynamic programming $\leadsto$

Theorem: Highly Connected Deletion can be solved in $O(3^{4k} \cdot k^2 + n^2 mk \cdot \log n)$ time.

FPT Algorithm and Further Data Reduction

Combination of branching, data reduction, and dynamic programming $\rightsquigarrow$

Theorem: Highly Connected Deletion can be solved in $O(3^{4k} \cdot k^2 + n^2 mk \cdot \log n)$ time.

Lemma: Let G be a highly connected graph. If two vertices in G are adjacent, they have at least one common neighbor.

FPT Algorithm and Further Data Reduction

Combination of branching, data reduction, and dynamic programming $\rightsquigarrow$

Theorem: Highly Connected Deletion can be solved in $O(3^{4k} \cdot k^2 + n^2 mk \cdot \log n)$ time.

Lemma: Let G be a highly connected graph. If two vertices in G are adjacent, they have at least one common neighbor.

$\rightsquigarrow$ **Reduction rule:** If there are two vertices that are connected by an edge but have no common neighbors, then delete the edge.

Highly Connected Deletion: PPI Experiments

	n	m	Δk	Δk [%]	n'	m'
C. elegans phys.	157	153	100	92.6	11	38
C. elegans all	3613	6828	5204	80.1	373	1562
M. musculus phys.	4146	7097	5659	85.3	426	1339
M. musculus all	5252	9640	7609	84.8	595	1893
A. thaliana phys.	1872	2828	2057	83.1	187	619
A. thaliana all	5704	12627	8797	79.5	866	3323

n', m': size of largest connected component after data reduction

Highly Connected Deletion: PPI Experiments

	n	m	Δk	Δk [%]	n'	m'
C. elegans phys.	157	153	100	92.6	11	38
C. elegans all	3613	6828	5204	80.1	373	1562
M. musculus phys.	4146	7097	5659	85.3	426	1339
M. musculus all	5252	9640	7609	84.8	595	1893
A. thaliana phys.	1872	2828	2057	83.1	187	619
A. thaliana all	5704	12627	8797	79.5	866	3323

n', m': size of largest connected component after data reduction

	min-cut without DR			min-cut with DR			column generation		
	k	s	t	k	s	t	k	s	t
CE-p	111	136	0.01	108	133	0.01	108	133	0.06
CE-a	6714	3589	86.46	6630	3521	6.36	6499	3436	2088.35
MM-p	7004	4116	126.30	6882	4003	7.42	6638	3845	898.13
MM-a	9563	5227	267.63	9336	5044	17.84	8978	4812	3858.62
AT-p	2671	1796	5.82	2567	1723	0.68	2476	1675	60.34
AT-a	12096	5559	434.52	11590	5213	32.09	11069	4944	34121.23

s: number of unclustered vertices; t: running time in seconds

Case Study: Graph Anonymization

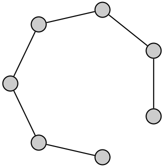
Degree Anonymization

Input: An undirected graph $G = (V, E)$ and two positive integers k and s.
Question: Is there an edge set E' over V with $|E'| \leq s$ such that $G' = (V, E \cup E')$ is *k-anonymous*, that is, for every vertex $v \in V$ there are at least $k - 1$ other vertices in G' having the same degree?

Case Study: Graph Anonymization

Degree Anonymization

Input: An undirected graph $G = (V, E)$ and two positive integers k and s.
Question: Is there an edge set E' over V with $|E'| \leq s$ such that $G' = (V, E \cup E')$ is *k-anonymous*, that is, for every vertex $v \in V$ there are at least $k - 1$ other vertices in G' having the same degree?

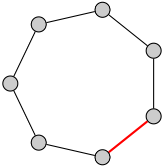


2-anonymous graph

Case Study: Graph Anonymization

Degree Anonymization

Input: An undirected graph $G = (V, E)$ and two positive integers k and s.
Question: Is there an edge set E' over V with $|E'| \leq s$ such that $G' = (V, E \cup E')$ is *k-anonymous*, that is, for every vertex $v \in V$ there are at least $k - 1$ other vertices in G' having the same degree?

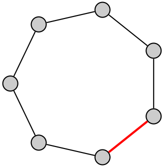


7-anonymous graph

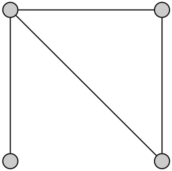
Case Study: Graph Anonymization

Degree Anonymization

Input: An undirected graph $G = (V, E)$ and two positive integers k and s.
Question: Is there an edge set E' over V with $|E'| \leq s$ such that $G' = (V, E \cup E')$ is *k-anonymous*, that is, for every vertex $v \in V$ there are at least $k - 1$ other vertices in G' having the same degree?



7-anonymous graph

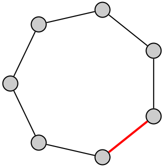


1-anonymous graph

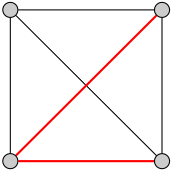
Case Study: Graph Anonymization

Degree Anonymization

Input: An undirected graph $G = (V, E)$ and two positive integers k and s.
Question: Is there an edge set E' over V with $|E'| \leq s$ such that $G' = (V, E \cup E')$ is *k-anonymous*, that is, for every vertex $v \in V$ there are at least $k - 1$ other vertices in G' having the same degree?



7-anonymous graph

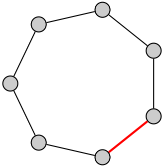


4-anonymous graph

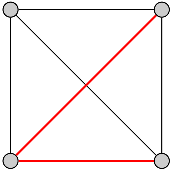
Case Study: Graph Anonymization

Degree Anonymization

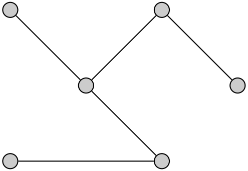
Input: An undirected graph $G = (V, E)$ and two positive integers k and s.
Question: Is there an edge set E' over V with $|E'| \leq s$ such that $G' = (V, E \cup E')$ is *k-anonymous*, that is, for every vertex $v \in V$ there are at least $k - 1$ other vertices in G' having the same degree?



7-anonymous graph



4-anonymous graph

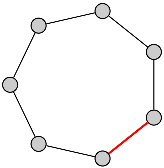


1-anonymous graph

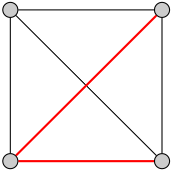
Case Study: Graph Anonymization

Degree Anonymization

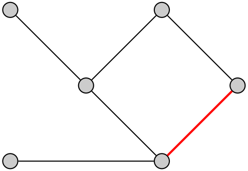
Input: An undirected graph $G = (V, E)$ and two positive integers k and s.
Question: Is there an edge set E' over V with $|E'| \leq s$ such that $G' = (V, E \cup E')$ is k*-anonymous*, that is, for every vertex $v \in V$ there are at least $k - 1$ other vertices in G' having the same degree?



7-anonymous graph

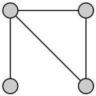


4-anonymous graph



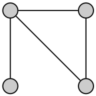
2-anonymous graph

Anonymization Heuristic, Liu and Terzi 2008



input graph

Anonymization Heuristic, Liu and Terzi 2008

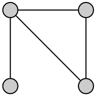


$\overset{1.}{\Rightarrow}$ 1,2,2,3

degree sequence

Step 1: Sorting the degrees.

Anonymization Heuristic, Liu and Terzi 2008

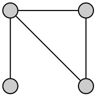


input graph $\overset{1.}{\Rightarrow}$ 1,2,2,3 (degree sequence) $\overset{2.}{\Rightarrow}$ 3,3,3,3 (“anonymized” degree sequence)

Step 1: Sorting the degrees.

Step 2: Standard dynamic programming (running time $O(n \cdot s \cdot k \cdot \Delta) = O(n^4)$).

Anonymization Heuristic, Liu and Terzi 2008



input graph

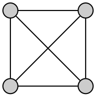
$\stackrel{1.}{\Rightarrow}$ 1,2,2,3

degree sequence

$\stackrel{2.}{\Rightarrow}$ 3,3,3,3

"anonymized" degree sequence

$\stackrel{3.}{\Rightarrow}$



"realized" degree sequence

Step 1: Sorting the degrees.

Step 2: Standard dynamic programming (running time $O(n \cdot s \cdot k \cdot \Delta) = O(n^4)$).

Step 3: (Due to graph structure not always possible!) If there exists a "realization", then it can be constructed in polynomial time ($\rightsquigarrow$ f-factors).

The Third Step of the Liu-Terzi-Heuristic

Lemma

If the solution (edge set) found in the dynamic programming is "large" (that is, $s > \Delta^4$ with Δ being the maximum vertex degree)), then there is always a realization of the anonymized degree sequence that is a supergraph of the input graph. This realization can be found in polynomial time.

The Third Step of the Liu-Terzi-Heuristic

Lemma

If the solution (edge set) found in the dynamic programming is “large” (that is, $s > \Delta^4$ with Δ being the maximum vertex degree)), then there is always a realization of the anonymized degree sequence that is a supergraph of the input graph. This realization can be found in polynomial time.

Consequence: win-win situation. Either

- the problem is polynomial-time solvable or
- the solution is “small” ($\leq \Delta^4$).

Case Study Incremental Conservative k-List Coloring

Incremental Clustering and Dynamic Information Retrieval
[Charikar, Chekuri, Feder, Motwani; STOC 1997; SICOMP 2004]:

$\rightsquigarrow$ **Incremental Clustering** problem:
For an update sequence of n points in metric space, maintain a collection of k clusters such that as each input point is presented, either it is assigned to one of the current k clusters or it starts off a new cluster while two existing clusters are merged into one.

Case Study Incremental Conservative k-List Coloring

Incremental Clustering and Dynamic Information Retrieval
[Charikar, Chekuri, Feder, Motwani; STOC 1997; SICOMP 2004]:

$\rightsquigarrow$ **Incremental Clustering** problem:
For an update sequence of n points in metric space, maintain a collection of k clusters such that as each input point is presented, either it is assigned to one of the current k clusters or it starts off a new cluster while two existing clusters are merged into one.

Remarks:

- Practial motivation: Maintain clusters in dynamic environments; updating clusterings without performing frequent reclustering is highly desirable.

Case Study Incremental Conservative k-List Coloring

Incremental Clustering and Dynamic Information Retrieval
[Charikar, Chekuri, Feder, Motwani; STOC 1997; SICOMP 2004]:

$\rightsquigarrow$ **Incremental Clustering** problem:
For an update sequence of n points in metric space, maintain a collection of k clusters such that as each input point is presented, either it is assigned to one of the current k clusters or it starts off a new cluster while two existing clusters are merged into one.

Remarks:

- Practial motivation: Maintain clusters in dynamic environments; updating clusterings without performing frequent reclustering is highly desirable.
- Closely related to *online* clustering model...
- Charikar et al. focus on polynomial-time approximation and investigate several variants of Incremental Clustering.

k-Center and k-List Coloring

k-Center

Input A distance function d on an element set X, $t \in \mathbb{R}^+$ and $k \in \mathbb{N}^+$.

Question Is there a k-partition $C_1, \ldots, C_k$ of X such that $\max_{1 \leq i \leq k} \max_{v,u \in C_i} d(v, u) \leq t$?

k-Center and k-List Coloring

k-Center

Input A distance function d on an element set X, $t \in \mathbb{R}^+$ and $k \in \mathbb{N}^+$.

Question Is there a k-partition $C_1, \ldots, C_k$ of X such that $\max_{1 \leq i \leq k} \max_{v,u \in C_i} d(v, u) \leq t$?

k-Center $\equiv_p$ k-List Coloring:

k-Center and k-List Coloring

k-Center

Input A distance function d on an element set X, $t \in \mathbb{R}^+$ and $k \in \mathbb{N}^+$.

Question Is there a k-partition $C_1, \ldots, C_k$ of X such that $\max_{1 \leq i \leq k} \max_{v,u \in C_i} d(v,u) \leq t$?

k-Center $\equiv_p$ k-List Coloring:

k-List Coloring

Input A graph $G = (V, E)$, $k \in \mathbb{N}^+$ and a list of colors $L(v) \subseteq \{1, \ldots, k\}$ for each $v \in V$.

Question Is there a k-list coloring $f : V \to \{1, 2, \ldots, k\}$ of G?

f is a **k-list coloring** of G iff $\forall u, v \in E : f(u) \neq f(v)$ and $\forall v \in V : f(v) \in L(v)$.

Incremental Conservative k-List Coloring

Incremental Conservative k-List Coloring (IC k-List Coloring)

Input A graph $G = (V, E)$, k-list coloring f for $G[V \setminus \{x\}]$, and number $c \in \mathbb{N}$ of allowed recolorings.

Question Is there a k-list coloring f' for G such that $|\{v \in V \setminus \{x\} \mid f(v) \neq f'(v)\}| \leq c$?

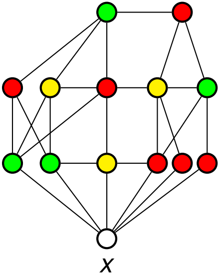
Incremental Conservative k-List Coloring

Incremental Conservative k-List Coloring (IC k-List Coloring)

Input A graph $G = (V, E)$, k-list coloring f for $G[V \setminus \{x\}]$, and number $c \in \mathbb{N}$ of allowed recolorings.

Question Is there a k-list coloring f' for G such that $|\{v \in V \setminus \{x\} \mid f(v) \neq f'(v)\}| \leq c$?

Example:



$k = 3$
$c = 3$
$L(v) \subseteq \{1, \ldots, k\} \; \forall v \in V$

Note: c measures degree of change allowed: *conservation* parameter.

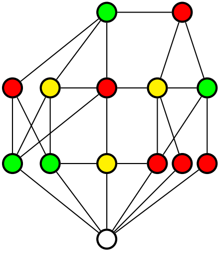
Incremental Conservative k-List Coloring

Incremental Conservative k-List Coloring (IC k-List Coloring)

Input A graph $G = (V, E)$, k-list coloring f for $G[V \setminus \{x\}]$, and number $c \in \mathbb{N}$ of allowed recolorings.

Question Is there a k-list coloring f' for G such that $|\{v \in V \setminus \{x\} \mid f(v) \neq f'(v)\}| \leq c$?

Example:



$k = 3$
$c = 3$
$L(v) \subseteq \{1, \ldots, k\}\ \forall v \in V$

Note: c measures degree of change allowed: *conservation* parameter.
Parameterize on conservation!

Complexity of IC k-List Coloring

Theorem. IC 3-Coloring is NP-complete even on bipartite graphs.

Idea of proof. Use corresponding NP-hardness result for "Precoloring Extension" due to
Bodlaender, Jansen, and Woeginger [Discrete Appl. Math. 1994]. □

Complexity of IC k-List Coloring

Theorem. IC 3-Coloring is NP-complete even on bipartite graphs.

Idea of proof. Use corresponding NP-hardness result for "Precoloring Extension" due to Bodlaender, Jansen, and Woeginger [Discrete Appl. Math. 1994]. □

Theorem. IC k-List Coloring is $W[1]$-hard with respect to the conservation parameter c.

Idea of proof. Parameterized reduction from k-Multicolored Independent Set. □

Complexity of IC k-List Coloring

Theorem. IC 3-Coloring is NP-complete even on bipartite graphs.

Idea of proof. Use corresponding NP-hardness result for "Precoloring Extension" due to Bodlaender, Jansen, and Woeginger [Discrete Appl. Math. 1994]. □

Theorem. IC k-List Coloring is $W[1]$-hard with respect to the conservation parameter c.

Idea of proof. Parameterized reduction from k-Multicolored Independent Set. □

$\rightsquigarrow$ Do a **multivariate** analysis! Combine parameters k and c:

Theorem. IC k-List Coloring can be solved in $O(k \cdot (k-1)^c \cdot |V|)$ time.

Idea of proof. Search tree algorithm with straightforward branching. □

IC *k*-List Coloring on Special Graphs

class	IC *k*-List Col.		IC *k*-Col.	PrExt	*k*-List Col.
	poly kernel				
trees	/	P	P	P	P
compl. bip.	?	NP*-c	P	**P**	**NP-c**
bipartite	no	NP-c	NP-c	**NP-c**	NP-c
chordal	?	NP*-c	NP-c	NP-c	NP-c
interval	?	NP*-c	?	**NP-c**	NP-c
unit interval	yes	NP*-c	?	**NP-c**	NP-c
cographs	?	?	?	**P**	**NP-c**
dist.-hered.	?	NP*-c	NP-c	**NP-c**	NP-c
split	?	NP*-c	?	**P**	**NP-c**

NP*-c: Turing reductions used; boldfaced results from literature.

IC k-List Coloring on Special Graphs

class	IC k-List Col. poly kernel		IC k-Col.	PrExt	k-List Col.
trees	/	P	P	P	P
compl. bip.	?	NP*-c	P	**P**	**NP-c**
bipartite	no	NP-c	NP-c	**NP-c**	NP-c
chordal	?	NP*-c	NP-c	NP-c	NP-c
interval	?	NP*-c	?	**NP-c**	NP-c
unit interval	yes	NP*-c	?	**NP-c**	NP-c
cographs	?	?	?	**P**	**NP-c**
dist.-hered.	?	NP*-c	NP-c	**NP-c**	NP-c
split	?	NP*-c	?	**P**	**NP-c**

NP*-c: Turing reductions used; boldfaced results from literature.

Note: Using dynamic programming, IC k-List Coloring can be solved in $O(k^{\omega+1}\omega^2 \cdot |V|)$ time on graphs of treewidth ω (with given tree dec.).

Experiments with IC k-list Coloring

Setting: Using the solution to IC k-List Coloring in a subroutine of a well-known **greedy** heuristic for graph coloring yields promising results.

Idea: If greedy (color according to descending vertex degree) fails, then try to conservatively recolor with $c \leq 8$.

Experiments with IC k-list Coloring

Setting: Using the solution to IC k-List Coloring in a subroutine of a well-known **greedy** heuristic for graph coloring yields promising results.

Idea: If greedy (color according to descending vertex degree) fails, then try to conservatively recolor with $c \leq 8$.

Compared with **Iterated Greedy** heuristic due to Culberson and Luo [DIMACS Series in Discrete Math. and Theor. Comput. Sci., 1996]: Iteratively run the greedy algorithm by trying different vertex orderings in the coloring process...; abort when after 1000 iterations no better coloring was found.

Experiments with IC k-list Coloring

Setting: Using the solution to IC k-List Coloring in a subroutine of a well-known **greedy** heuristic for graph coloring yields promising results.

Idea: If greedy (color according to descending vertex degree) fails, then try to conservatively recolor with $c \leq 8$.

Compared with **Iterated Greedy** heuristic due to Culberson and Luo [DIMACS Series in Discrete Math. and Theor. Comput. Sci., 1996]: Iteratively run the greedy algorithm by trying different vertex orderings in the coloring process...; abort when after 1000 iterations no better coloring was found.

Tested on 64 benchmark graph instances (between 25 and 4730 vertices and 15 % average edge density), taken from
DIMACS "Graph Coloring and its Generalizations" Symposium 2002.

IC *k*-List Coloring Experimental Work II

Each value obtained as avg. over four runs (std. deviation in brackets):

	greedy		Iterated Greedy			search tree		
	k	time	k	#iter	time	k	c	time
gr1	8	0.2	5.0 [0.0]	1058.8	33.2 [1.3]	5.0 [0.0]	8	0.2 [0.0]
gr2	29	0.1	27.5 [0.6]	1243.8	24.5 [5.0]	25.0 [0.0]	6	0.5 [0.1]
gr3	17	0.0	16.8 [0.5]	1217.5	6.7 [2.3]	14.8 [0.5]	8	0.3 [0.1]
gr4	148	0.3	109 [1.4]	2765.3	68.8 [9.2]	116.8 [2.6]	4	1.5 [0.6]
gr5	18	0.0	18.0 [0.0]	1000	5.0 [0.0]	16.0 [0.0]	7	1.2 [0.2]
gr6	44	0.3	42.0 [0.0]	1033.8	59.9 [1.7]	41.0 [0.0]	5	4.8 [0.3]
gr7	26	0.0	20.3 [0.5]	1295	2.2 [0.7]	19.3 [0.5]	7	0.4 [0.2]
gr8	31	0.0	14.0 [0.0]	1443.8	3.7 [0.3]	23.0 [5.4]	6	0.2 [0.1]
gr9	55	4.0	53.8 [0.5]	1006.3	475.5 [3.4]	50.0 [0.0]	5	3.9 [0.3]

IC *k*-List Coloring Experimental Work II

Each value obtained as avg. over four runs (std. deviation in brackets):

	greedy		Iterated Greedy			search tree		
	k	time	k	#iter	time	k	c	time
gr1	8	0.2	5.0 [0.0]	1058.8	33.2 [1.3]	5.0 [0.0]	8	0.2 [0.0]
gr2	29	0.1	27.5 [0.6]	1243.8	24.5 [5.0]	25.0 [0.0]	6	0.5 [0.1]
gr3	17	0.0	16.8 [0.5]	1217.5	6.7 [2.3]	14.8 [0.5]	8	0.3 [0.1]
gr4	148	0.3	109 [1.4]	2765.3	68.8 [9.2]	116.8 [2.6]	4	1.5 [0.6]
gr5	18	0.0	18.0 [0.0]	1000	5.0 [0.0]	16.0 [0.0]	7	1.2 [0.2]
gr6	44	0.3	42.0 [0.0]	1033.8	59.9 [1.7]	41.0 [0.0]	5	4.8 [0.3]
gr7	26	0.0	20.3 [0.5]	1295	2.2 [0.7]	19.3 [0.5]	7	0.4 [0.2]
gr8	31	0.0	14.0 [0.0]	1443.8	3.7 [0.3]	23.0 [5.4]	6	0.2 [0.1]
gr9	55	4.0	53.8 [0.5]	1006.3	475.5 [3.4]	50.0 [0.0]	5	3.9 [0.3]

Findings:
Our conservative search tree algorithm improves greedy result in 89 % of the tested instances, Iterated Greedy in 83 %. Improvement by 12 % resp. 11 % of number of colors used.

IC *k*-List Coloring Experimental Work II

Each value obtained as avg. over four runs (std. deviation in brackets):

	greedy		Iterated Greedy			search tree		
	k	time	k	#iter	time	k	c	time
gr1	8	0.2	5.0 [0.0]	1058.8	33.2 [1.3]	5.0 [0.0]	8	0.2 [0.0]
gr2	29	0.1	27.5 [0.6]	1243.8	24.5 [5.0]	25.0 [0.0]	6	0.5 [0.1]
gr3	17	0.0	16.8 [0.5]	1217.5	6.7 [2.3]	14.8 [0.5]	8	0.3 [0.1]
gr4	148	0.3	109 [1.4]	2765.3	68.8 [9.2]	116.8 [2.6]	4	1.5 [0.6]
gr5	18	0.0	18.0 [0.0]	1000	5.0 [0.0]	16.0 [0.0]	7	1.2 [0.2]
gr6	44	0.3	42.0 [0.0]	1033.8	59.9 [1.7]	41.0 [0.0]	5	4.8 [0.3]
gr7	26	0.0	20.3 [0.5]	1295	2.2 [0.7]	19.3 [0.5]	7	0.4 [0.2]
gr8	31	0.0	14.0 [0.0]	1443.8	3.7 [0.3]	23.0 [5.4]	6	0.2 [0.1]
gr9	55	4.0	53.8 [0.5]	1006.3	475.5 [3.4]	50.0 [0.0]	5	3.9 [0.3]

Findings:
Our conservative search tree algorithm improves greedy result in 89 % of the tested instances, Iterated Greedy in 83 %. Improvement by 12 % resp. 11 % of number of colors used.
Search tree algorithm by a factor of 50 slower than greedy algorithm, Iterative Greedy by a factor of 170.

Discussion and Outlook

- Every problem accompanied with a natural parameter space (that is, structure) to navigate through.
  $\rightsquigarrow$ Relevance of investigating the combinatorial relationships between different parameters and their combinations.

Discussion and Outlook

- Every problem accompanied with a natural parameter space (that is, structure) to navigate through.
  $\rightsquigarrow$ Relevance of investigating the combinatorial relationships between different parameters and their combinations.
- Structure analysis connects very well with multivariate algorithmics.

Discussion and Outlook

- Every problem accompanied with a natural parameter space (that is, structure) to navigate through.
  $\rightsquigarrow$ Relevance of investigating the combinatorial relationships between different parameters and their combinations.
- Structure analysis connects very well with multivariate algorithmics.
- Potential practical relevance when taking into account structure occurring in real-world data.
  $\rightsquigarrow$ Algorithm engineering through parameter identification in real-world instances.

Discussion and Outlook

- Every problem accompanied with a natural parameter space (that is, structure) to navigate through.
  $\rightsquigarrow$ Relevance of investigating the combinatorial relationships between different parameters and their combinations.
- Structure analysis connects very well with multivariate algorithmics.
- Potential practical relevance when taking into account structure occurring in real-world data.
  $\rightsquigarrow$ Algorithm engineering through parameter identification in real-world instances.
- Detecting "hidden parameters" may help in better understanding and exploiting the power of heuristics.

Discussion and Outlook

- Every problem accompanied with a natural parameter space (that is, structure) to navigate through.
  $\rightsquigarrow$ Relevance of investigating the combinatorial relationships between different parameters and their combinations.
- Structure analysis connects very well with multivariate algorithmics.
- Potential practical relevance when taking into account structure occurring in real-world data.
  $\rightsquigarrow$ Algorithm engineering through parameter identification in real-world instances.
- Detecting "hidden parameters" may help in better understanding and exploiting the power of heuristics.
- Potential drawback from a practical point of view: all our studies still rely on worst-case analysis.

Discussion and Outlook

- Every problem accompanied with a natural parameter space (that is, structure) to navigate through.
  $\rightsquigarrow$ Relevance of investigating the combinatorial relationships between different parameters and their combinations.
- Structure analysis connects very well with multivariate algorithmics.
- Potential practical relevance when taking into account structure occurring in real-world data.
  $\rightsquigarrow$ Algorithm engineering through parameter identification in real-world instances.
- Detecting "hidden parameters" may help in better understanding and exploiting the power of heuristics.
- Potential drawback from a practical point of view: all our studies still rely on worst-case analysis.
- Once a whole suite of (parameterized) algorithms is available, choosing the "right" one depending on the current input data becomes more relevant...

References

- Related survey: C. Komusiewicz, R. Niedermeier: New Races in Parameterized Algorithmics. MFCS 2012.
- Highly Connected Deletion: F. Hüffner, C. Komusiewicz, A. Liebtrau, R. Niedermeier. Partitioning Biological Networks into Highly Connected Clusters with Maximum Edge Coverage. ISBRA 2013.
- Graph Anonymization: S. Hartung, A. Nichterlein, R. Niedermeier, O. Suchý. A Refined Complexity Analysis of Degree Anonymization on Graphs. ICALP 2013.
- Incremental Conservative k-List Coloring: S. Hartung and R. Niedermeier. Incremental list coloring of graphs, parameterized by conservation. Theoretical Computer Science 2013.