Incremental List Coloring of Graphs, Parameterized by Conservation[2]

Sepp Hartung and Rolf Niedermeier

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

JMM 2013, San Diego

[2] Paper to appear in *Theoretical Computer Science*.

Leitmotif: Evolution Instead of Revolution!

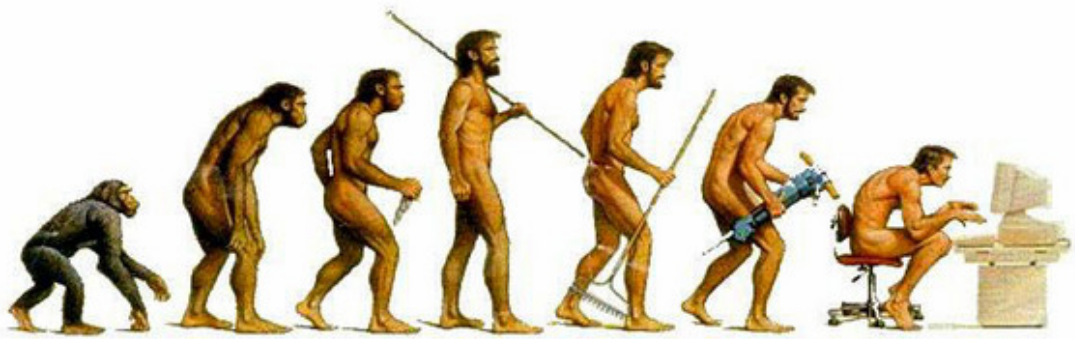
No to

but **yes** to

Leitmotif: Evolution Instead of Revolution!

No to

but **yes** to



Outline of the Talk

Basic contents:

- Parameterized local search in terms of "conservative parameterization" for graph (list) coloring.
- Proof of concept for practical relevance through preliminary experiments in a heuristic graph coloring context.

Outline of the Talk

Basic contents:

- Parameterized local search in terms of "conservative parameterization" for graph (list) coloring.
- Proof of concept for practical relevance through preliminary experiments in a heuristic graph coloring context.

Used Methods:

Nothing really new! Standard tools from parameterized complexity analysis, including hardness, kernelization and no-poly-kernel results, search trees, and dynamic programming.

Outline of the Talk

Basic contents:

- Parameterized local search in terms of "conservative parameterization" for graph (list) coloring.
- Proof of concept for practical relevance through preliminary experiments in a heuristic graph coloring context.

Used Methods:
Nothing really new! Standard tools from parameterized complexity analysis, including hardness, kernelization and no-poly-kernel results, search trees, and dynamic programming.

Basic message:
Desire for **evolution instead of revolution** provides opportunities for research on parameterized and multivariate complexity analysis.

Motivation, Stream 1: (Parameterized) Local Search

A famous example:

Travelling Salesperson Problem (TSP)

Input An edge-weighted graph G.

Task Find a minimum weight Hamiltonian cycle in G.

Motivation, Stream 1: (Parameterized) Local Search

A famous example:

Travelling Salesperson Problem (TSP)

Input An edge-weighted graph G.

Task Find a minimum weight Hamiltonian cycle in G.

Local search variant:

Local TSP

Input An edge-weighted graph $G = (V, E)$, an integer k, and a labeling such that $id = v_1, v_2, \ldots, v_n, v_1$ is a Hamiltonian cycle in G.

Question Is there an improved Hamiltonian cycle within the k-neighborhood of id in G?

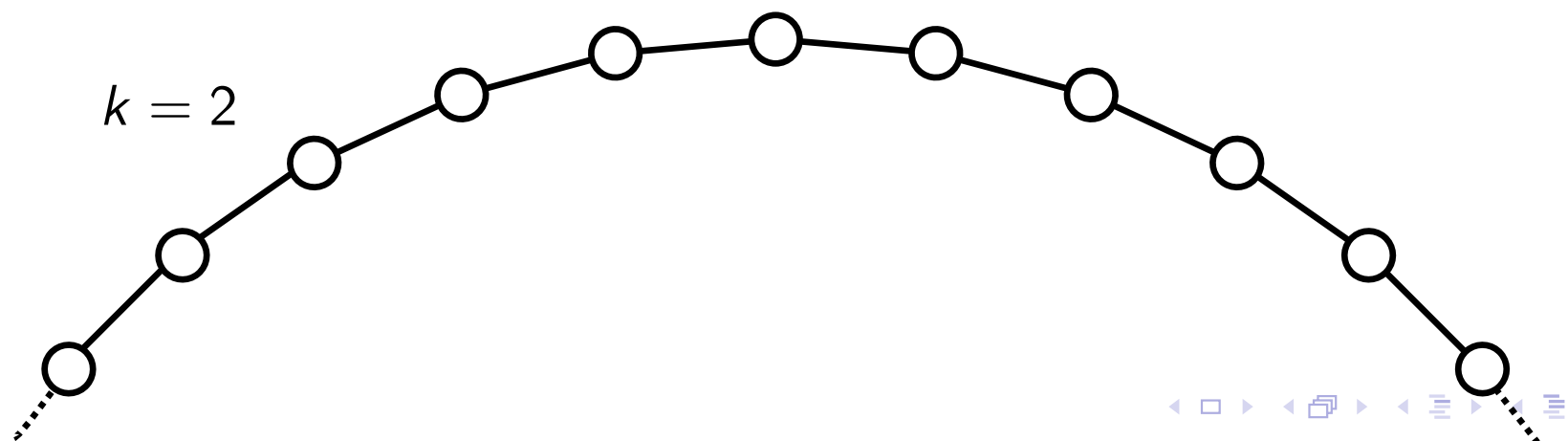
Motivation, Stream 1: Local TSP I

Local TSP

Input An edge-weighted graph $G = (V, E)$, an integer k, and a labeling such that $id = v_1, v_2, \ldots, v_n, v_1$ is a Hamiltonian cycle in G.

Question Is there an improved Hamiltonian cycle within the k-neighborhood of id in G?

How to define the k-neighborhood of a Hamiltonian cycle?
$\rightsquigarrow$ k-**Edge** neighborhood: exchange at most k edges of id.



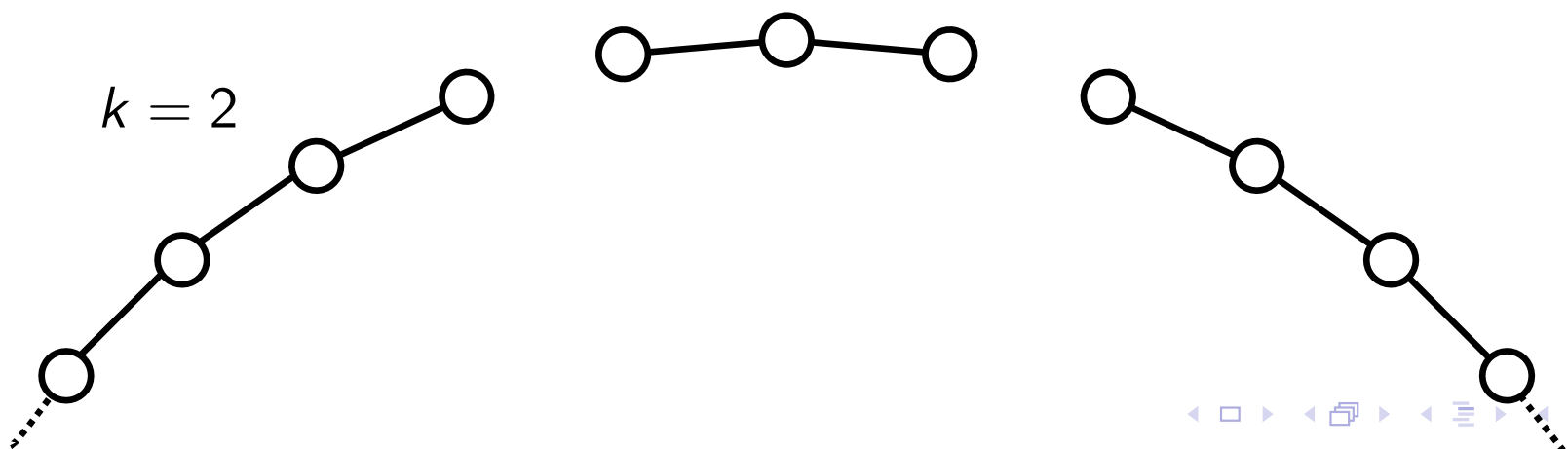
Motivation, Stream 1: Local TSP I

Local TSP

Input An edge-weighted graph $G = (V, E)$, an integer k, and a labeling such that $id = v_1, v_2, \ldots, v_n, v_1$ is a Hamiltonian cycle in G.

Question Is there an improved Hamiltonian cycle within the k-neighborhood of id in G?

How to define the k-neighborhood of a Hamiltonian cycle?
$\rightsquigarrow$ k-**Edge** neighborhood: exchange at most k edges of id.



Motivation, Stream 1: Local TSP I

Local TSP

Input An edge-weighted graph $G = (V, E)$, an integer k, and a labeling such that $id = v_1, v_2, \ldots, v_n, v_1$ is a Hamiltonian cycle in G.

Question Is there an improved Hamiltonian cycle within the k-neighborhood of id in G?

How to define the k-neighborhood of a Hamiltonian cycle?
$\rightsquigarrow$ k-**Edge** neighborhood: exchange at most k edges of id.

Running time for Local TSP(Edge)?

- Straightforward brute-force gives $n^{k+O(1)}$ running time.
- Question: $\exists\ f(k) \cdot n^{O(1)}$-time algorithm (FPT)?
  - No! $\rightarrow$ Marx [Oper. Res. Letters 2008]:
    Searching the k-change neighborhood for TSP is W[1]-hard.

Motivation, Stream 1: Local TSP II

Neighborhood measures besides edge neighborhood:

Max-Shift: Move every vertex at most k positions apart.
$\rightsquigarrow$ Balas [Annals of OR 1999]:
$O(4^{k-1} \cdot k^{1.5} \cdot n)$ time algorithm for Local TSP(Max-Shift)

Motivation, Stream 1: Local TSP II

Neighborhood measures besides edge neighborhood:

Max-Shift: Move every vertex at most k positions apart.
$\rightsquigarrow$ Balas [Annals of OR 1999]:
$O(4^{k-1} \cdot k^{1.5} \cdot n)$ time algorithm for Local TSP(Max-Shift)

Further measures of interest:
For $id = 1, 2, \ldots, n$, define the following operations:

- swap(i, j) results in $1, \ldots, i-1, j, i+1, \ldots, j-1, i, j+1, \ldots, n$;
- edit(i, j) results in $1, \ldots, i-1, i+1, \ldots, j-1, j, i, j+1, \ldots, n$;
- reversal(i, j) results in $1, \ldots, i-1, j, j-1, \ldots, i+1, i, j+1, \ldots, n$.

Motivation, Stream 1: Local TSP II

Neighborhood measures besides edge neighborhood:

Max-Shift: Move every vertex at most k positions apart.
$\rightsquigarrow$ Balas [Annals of OR 1999]:
$O(4^{k-1} \cdot k^{1.5} \cdot n)$ time algorithm for Local TSP(Max-Shift)

Further measures of interest:
For $id = 1, 2, \ldots, n$, define the following operations:

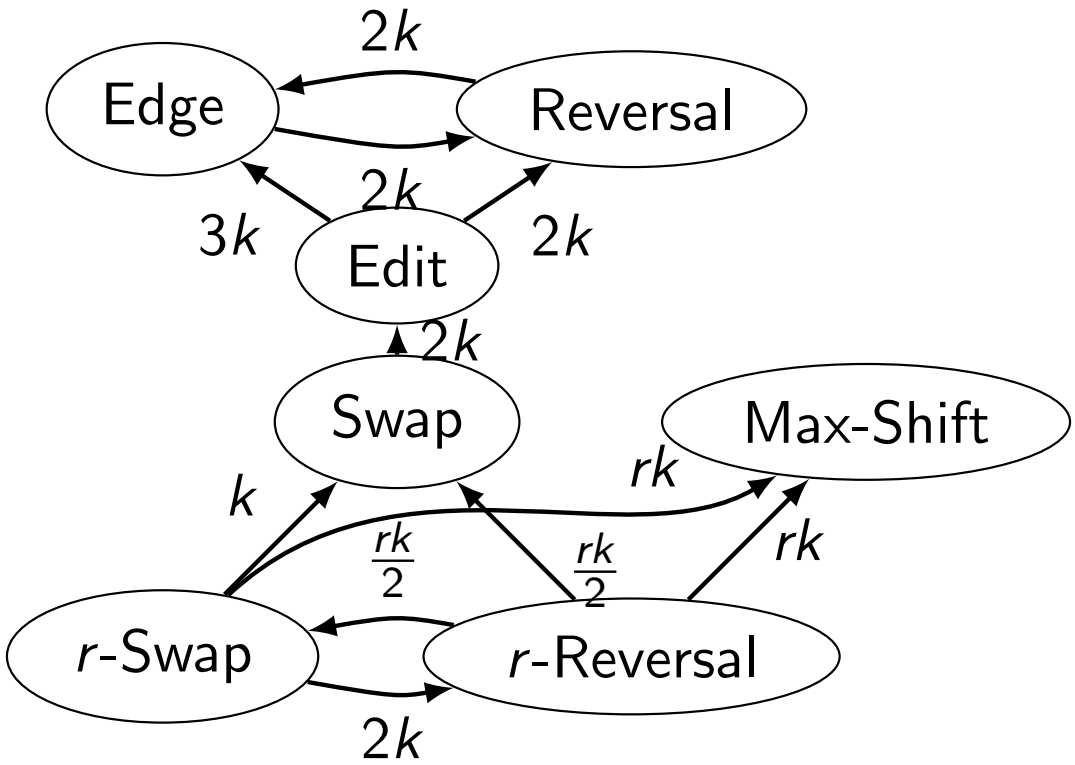
- swap(i, j) results in $1, \ldots, i-1, j, i+1, \ldots, j-1, i, j+1, \ldots, n$;
- edit(i, j) results in $1, \ldots, i-1, i+1, \ldots, j-1, j, i, j+1, \ldots, n$;
- reversal(i, j) results in $1, \ldots, i-1, j, j-1, \ldots, i+1, i, j+1, \ldots, n$.

Applying at most k operations leads to **Swap**, **Edit**, and **Reversal** k-neighborhood measures.

(Moreover, $0 < j - i < r \rightsquigarrow$ r-**Swap** and r-**Reversal**.)

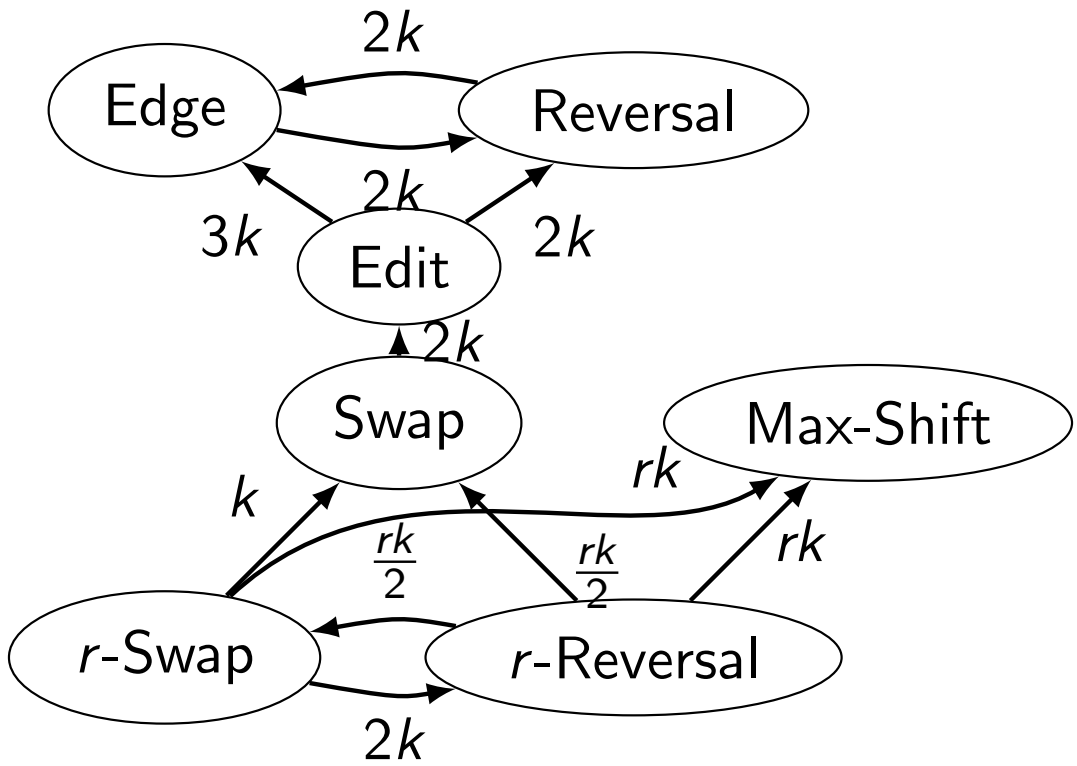
Motivation, Stream 1: Local TSP III

Parameter space of Local TSP distance measures, partially ordered [Guo, Hartung, N., Suchý; ISAAC 2011, Algorithmica online available]:



Motivation, Stream 1: Local TSP III

Parameter space of Local TSP distance measures, partially ordered [Guo, Hartung, N., Suchý; ISAAC 2011, Algorithmica online available]:



graphs	Swap	r-Swap	Edit	Reversal	r-Reversal	Edge
general	W[1]-h	FPT	W[1]-h	W[1]-h	FPT	W[1]-h
planar	FPT	FPT	FPT	?	FPT	?

Motivation, Stream 2: (Parameterized) Incrementalization

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.

Motivation, Stream 2: (Parameterized) Incrementalization

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.

Motivation, Stream 2: (Parameterized) Incrementalization

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, but incremental model is more restrictive (adversary does not need to respect input order...).
- Charikar et al. focus on polynomial-time approximation and investigate several variants of Incremental Clustering.

Motivation, Stream 2: 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 $\max_{1 \leq i \leq k} \max_{v,u \in C_i} d(v,u) \leq t$?

Motivation, Stream 2: 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:

Motivation, Stream 2: 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 $\forall v \in V : f(v) \in L(v)$.

Motiv., Stream 2: 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 $|\{v \in V \setminus \{x\} \mid f(v) \neq f'(v)\}| \leq c$?

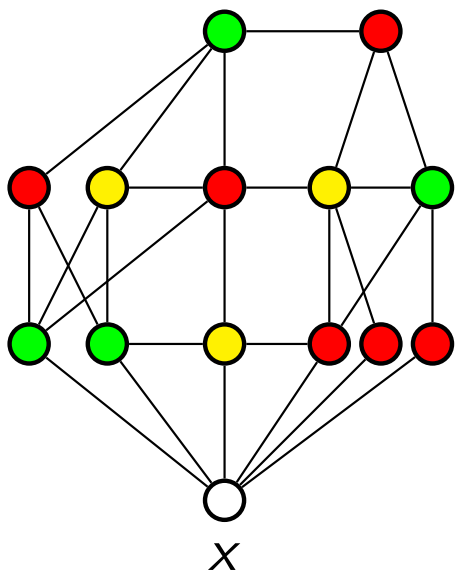
Motiv., Stream 2: Incremental Conservative k-List Coloring

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.

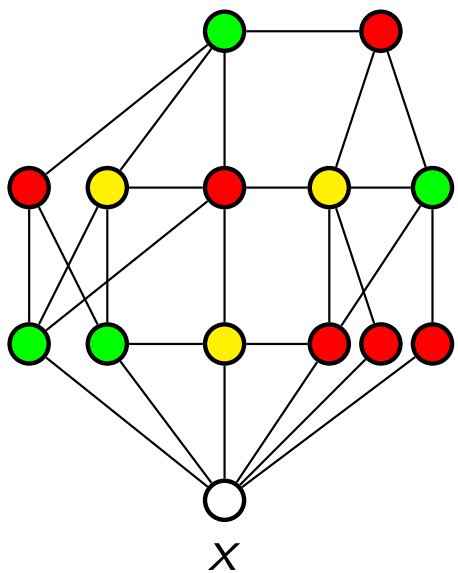
Motiv., Stream 2: Incremental Conservative k-List Coloring

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!

(Incremental Conservative) k-List Coloring I

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]. □

(Incremental Conservative) k-List Coloring I

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. □

(Incremental Conservative) k-List Coloring I

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. □

(Incremental Conservative) k-List Coloring I

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. □

Kernelization for IC k-List Coloring I

Theorem. IC k-List Coloring

1. admits a $(c \cdot k \cdot c! \cdot (k-1)^{c-1})$-vertex kernel, which can be computed in $O(|V| + |E|)$ time;

Kernelization for IC k-List Coloring I

Theorem. IC k-List Coloring

1. admits a $(c \cdot k \cdot c! \cdot (k-1)^{c-1})$-vertex kernel, which can be computed in $O(|V| + |E|)$ time;
2. admits a $3(k-1)^c$-vertex kernel, which can be computed in $O(|V| \log |V| + |E|)$ time;

Kernelization for IC k-List Coloring I

Theorem. IC k-List Coloring

1. admits a $(c \cdot k \cdot c! \cdot (k-1)^{c-1})$-vertex kernel, which can be computed in $O(|V| + |E|)$ time;
2. admits a $3(k-1)^c$-vertex kernel, which can be computed in $O(|V| \log |V| + |E|)$ time;
3. does not admit a polynomial kernel, unless NP $\subseteq$ coNP/poly.

Ideas of proofs.

Idea of kernelization:

- Compute a shortest "**possible conflict path**" P_v for $v \in V$.
- If length of P_v is $> c$, then v can be removed (too expensive to recolor v; update color lists of neighbors correspondingly by deleting v's color from these).

Kernelization for IC k-List Coloring I

Theorem. IC k-List Coloring

1. admits a $(c \cdot k \cdot c! \cdot (k-1)^{c-1})$-vertex kernel, which can be computed in $O(|V| + |E|)$ time;
2. admits a $3(k-1)^c$-vertex kernel, which can be computed in $O(|V| \log |V| + |E|)$ time;
3. does not admit a polynomial kernel, unless NP $\subseteq$ coNP/poly.

Ideas of proofs.

Idea of kernelization:

- Compute a shortest "**possible conflict path**" P_v for $v \in V$.
- If length of P_v is $> c$, then v can be removed (too expensive to recolor v; update color lists of neighbors correspondingly by deleting v's color from these).

Idea of no-poly-kernel result:
"Standard use" of Bodlaender, Downey, Fellows, and Hermelin's framework [JCSS 2009]. Show "OR-compositionality".

Kernelization for IC k-List Coloring II

Let S be set of $\leq c+1$ recolored vertices, including x.
W.l.o.g. $G[S]$ is connected.

Conflict path for $v_j \in S$: path $[x, v_1, \ldots, v_j]$ such that $f'(x) = f(v_1)$ and $f'(v_i) = f(v_{i+1})\ \forall 1 \leq i < j$

Kernelization for IC k-List Coloring II

Let S be set of $\leq c+1$ recolored vertices, including x.
W.l.o.g. $G[S]$ is connected.

Conflict path for $v_j \in S$: path $[x, v_1, \ldots, v_j]$ such that $f'(x) = f(v_1)$ and $f'(v_i) = f(v_{i+1})$ $\forall 1 \leq i < j$

Lemma. $G[S]$ contains a length-"$\leq c$" conflict path for every $v \in S$.

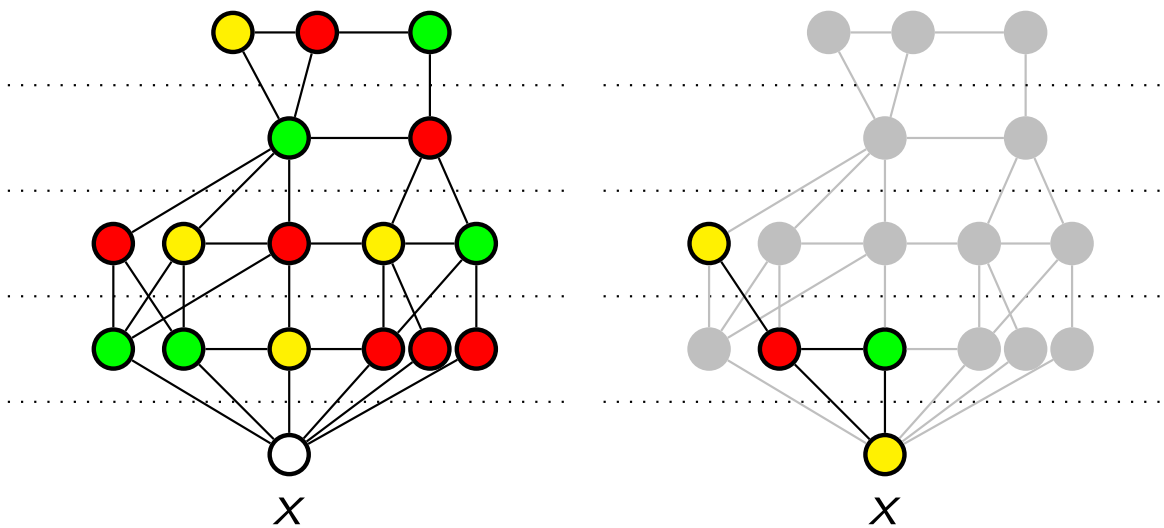
Kernelization for IC k-List Coloring II

Let S be set of $\leq c+1$ recolored vertices, including x.
W.l.o.g. $G[S]$ is connected.

Conflict path for $v_j \in S$: path $[x, v_1, \ldots, v_j]$ such that $f'(x) = f(v_1)$ and $f'(v_i) = f(v_{i+1})$ $\forall 1 \leq i < j$

Lemma. $G[S]$ contains a length-"$\leq c$" conflict path for every $v \in S$.

Example with $c = 3$:



Kernelization for IC k-List Coloring II

Lemma. $G[S]$ contains a length-"$\leq c$" conflict path to every $v \in S$.

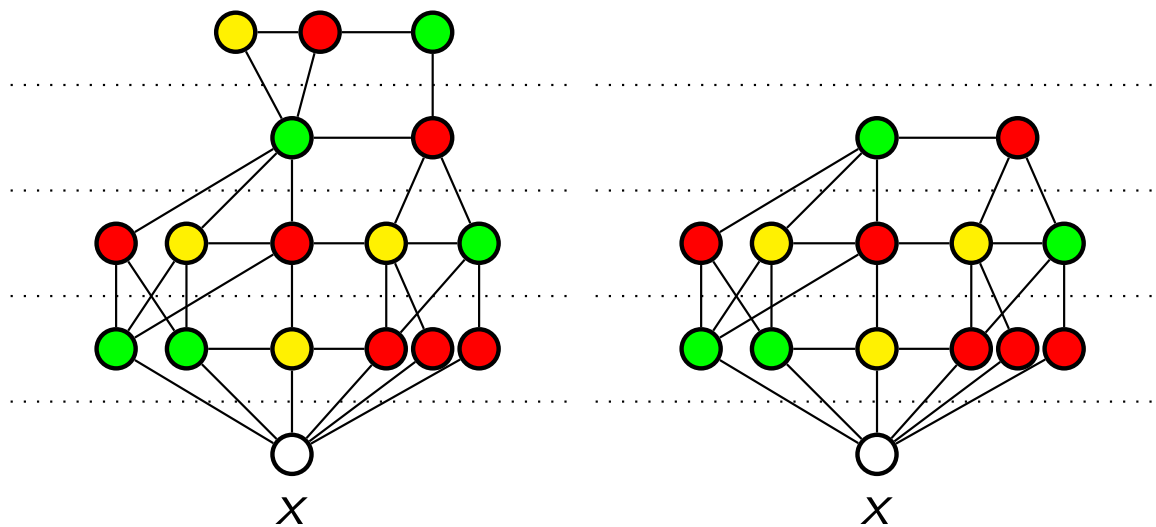
$\rightsquigarrow$ **Reduction rule:** Remove vertices with length-"$> c$" conflict paths...

Kernelization for IC k-List Coloring II

Lemma. $G[S]$ contains a length-"$\leq c$" conflict path to every $v \in S$.

$\rightsquigarrow$ **Reduction rule:** Remove vertices with length-"$> c$" conflict paths...

Example with $c = 3$**:**



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-List Col.	PrExt	k-List Col.
trees	/	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.).

Proof of Concept: Experimental Work I

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 $c \leq 8$.

Proof of Concept: Experimental Work I

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.

Proof of Concept: Experimental Work I

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.

Proof of Concept: 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]

Proof of Concept: 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.

Proof of Concept: 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.

Concluding Remarks

- Several open problems for various graph classes (see table).

Concluding Remarks

- Several open problems for various graph classes (see table).
- Our studies just focussed on IC k-List Coloring; numerous other problems (particularly clustering problems due to strong motivation) remain to be studied.

Concluding Remarks

- Several open problems for various graph classes (see table).
- Our studies just focussed on IC k-List Coloring; numerous other problems (particularly clustering problems due to strong motivation) remain to be studied.
- Investigate connection to related fields of "reoptimization" of locally modified instances of optimization problems and "minimal cost reconfigurations" in data migration scenarios (distributed servers such as in video on demand).

Concluding Remarks

- Several open problems for various graph classes (see table).
- Our studies just focussed on IC k-List Coloring; numerous other problems (particularly clustering problems due to strong motivation) remain to be studied.
- Investigate connection to related fields of "reoptimization" of locally modified instances of optimization problems and "minimal cost reconfigurations" in data migration scenarios (distributed servers such as in video on demand).
- Study more general settings, e.g. with more than one vertex added, vertex deletions, etc.

Evolution is Conservative—Changes Little!