Some Results About Graph Contraction Problems.

Prafullkumar Tale

25th May, 2020

Graph Modification Problems

$\mathcal{G}$ – Graph class (poly time recognisable)

Graph Modification Problems

$\mathcal{G}$ – Graph class (poly time recognisable)

Input: Graph G, int k

Graph Modification Problems

$\mathcal{G}$ – Graph class (poly time recognisable)

Input: Graph G, int k

Question: Can we make at most k modifications in G s.t. resulting graph is in $\mathcal{G}$?

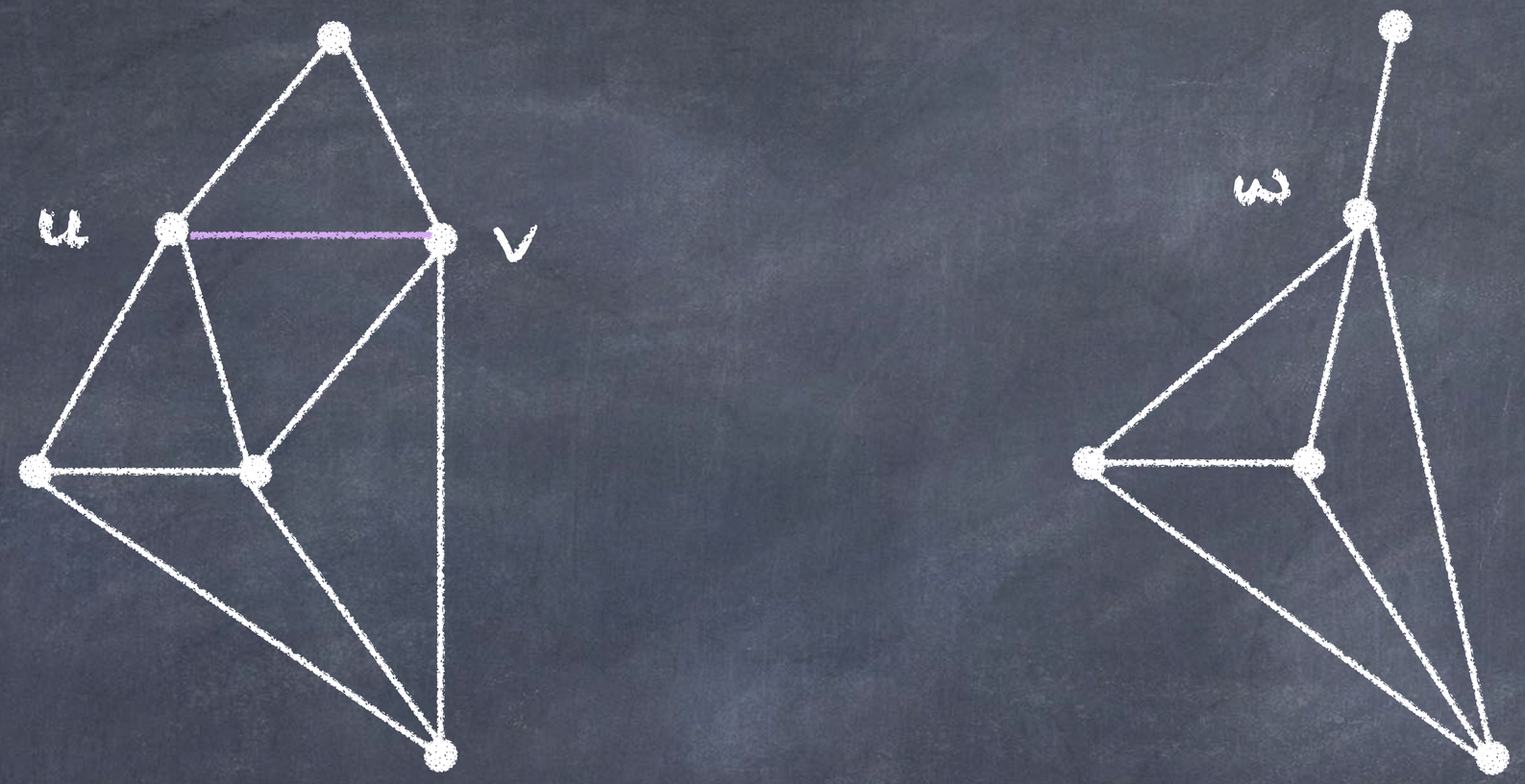
Graph Modification Problems

$\mathcal{G}$ – Graph class (poly time recognisable)

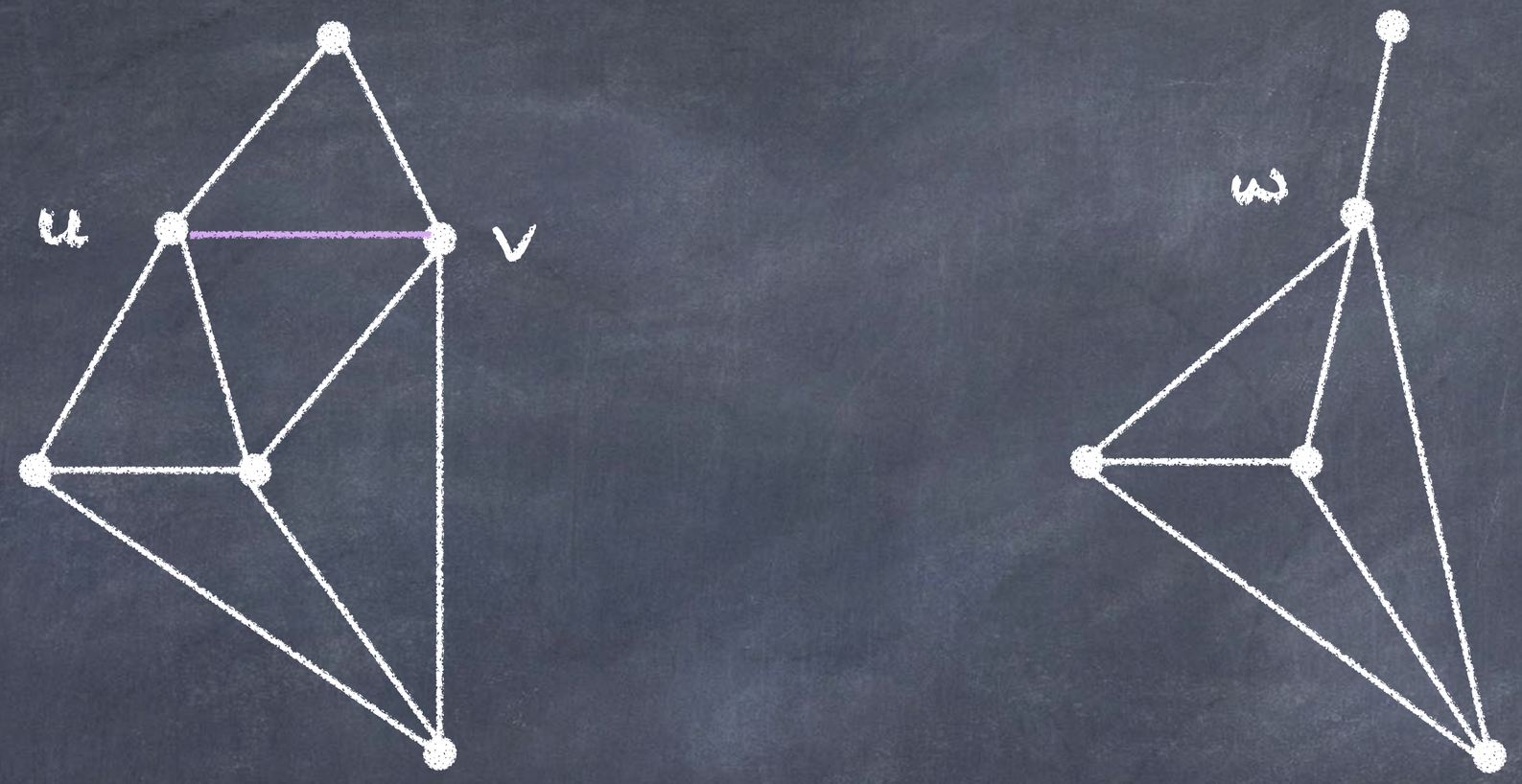
Input: Graph G, int k

Question: Can we make at most k modifications in G s.t. resulting graph is in $\mathcal{G}$?

Modification : (1) vertex deletion (2) edge deletion (3) edge addition (4) edge contraction



Edge contraction



Edge contraction

No self-loop
No parallel edges

$\mathcal{G}$ – Contraction

Input: Graph G, int k

Question: Can we contract at most k edges in G s.t. resulting graph is in $\mathcal{G}$?

Parameterized Complexity and Known Results

Overview Of Our Contributions

Brief description: Lossy Kernel for Clique Cont.

Parameterized Complexity and Known Results

Overview Of Our Contributions

Brief description: Lossy Kernel for Clique Cont.

Parameterized Complexity

Parameterized Complexity

- Instance + relevant secondary measure (i.e. parameter)

  Ex. For $\mathcal{G}$-Contraction Problem
  Instance: (G, k)
  Parameter: k

Parameterized Complexity

- Instance + relevant secondary measure (i.e. parameter)

  Ex. For $\mathcal{G}$-Contraction Problem
  Instance: (G, k)
  Parameter: k

- Objective : Find an algorithm f(k) poly(n)

  f(k) – function depending only on k
  k – collection of parameters

- Fixed Parameter Tractable (FPT): Class of problems that can be solved in f(k) poly(n).

n – size of input
k – parameter

- Fixed Parameter Tractable (FPT): Class of problems that can be solved in f(k) poly(n).

  n – size of input
  k – parameter

- Such algorithm may not exists for every problem + selected parameter W[i]-Hardness

- Fixed Parameter Tractable (FPT): Class of problems that can be solved in f(k) poly(n).

  n – size of input
  k – parameter

- Such algorithm may not exists for every problem + selected parameter W[i]-Hardness

Redefining FPT : Kernelization

Problem admits a kernel of size $g(k)$ if there exists an poly-time algorithm that

Redefining FPT : Kernelization

Problem admits a kernel of size $g(k)$ if there exists an poly-time algorithm that

– on input (G, k) produces output (G', k')
s.t. $|G'| + k' <= g(k)$

Redefining FPT : Kernelization

Problem admits a kernel of size $g(k)$ if there exists an poly-time algorithm that

- on input (G, k) produces output (G', k') s.t. $|G'| + k' <= g(k)$
- (G, k) is a Yes instance if and only if (G', k') is.

Redefining FPT : Kernelization

Problem admits a kernel of size $g(k)$ if there exists an poly-time algorithm that

- on input (G, k) produces output (G', k') s.t. $|G'| + k' \leq g(k)$
- (G, k) is a Yes instance if and only if (G', k') is.

Theorem : Problem is in FPT iff it admits a kernel.

Theorem : Problem is in FPT iff it admits a kernel.

Theorem : Problem is in FPT iff it admits a kernel.

Theorem : Problem is in FPT iff it admits a kernel.

Theorem : Problem is in FPT iff it admits a kernel.

Lower Bounds unless NP $\subseteq$ coNP/poly

Theorem : Problem is in FPT iff it admits a kernel.

Lower Bounds unless NP $\subseteq$ coNP/poly

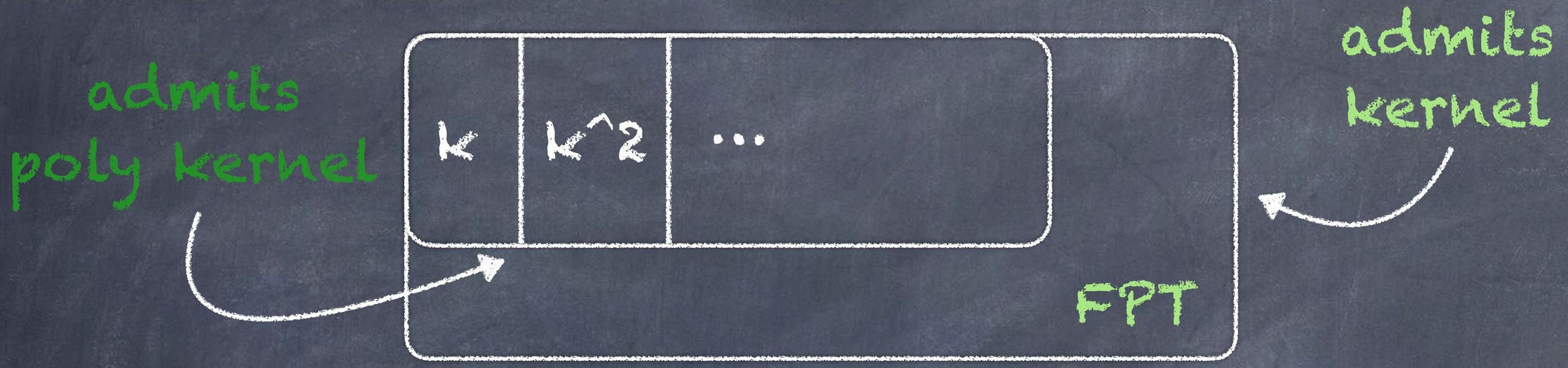
- For problem in FPT: it does not admit a poly kernel.

Theorem : Problem is in FPT iff it admits a kernel.

Lower Bounds unless NP $\subseteq$ coNP/poly

- For problem in FPT: it does not admit a poly kernel.
- For problem admitting poly-kernel: it does not admit a kernel of size $k^{c-\epsilon}$, for some constant c and $\epsilon > 0$.

Theorem : Problem is in FPT iff it admits a kernel.



Lower Bounds unless NP $\subseteq$ coNP/poly

- For problem in FPT: it does not admit a poly kernel.
- For problem admitting poly-kernel: it does not admit a kernel of size $k^{c-\epsilon}$, for some constant c and $\epsilon > 0$.

$\mathcal{G}$ – Contraction: FPT

$\mathcal{G}$ – Contraction: FPT

Heggernes + H.L.L.P. (2011): Path | Tree exp(k)

$\mathcal{G}$ – Contraction: FPT

Heggernes + H.L.L.P. (2011):	Path | Tree	exp(k)
Golovach + K.P.T. (2011):	min-deg >= 14	exp(k)

$\mathcal{G}$ – Contraction: FPT

Heggernes + H.L.L.P. (2011):	Path | Tree	exp(k)
Golovach + K.P.T. (2011):	min-deg >= 14	exp(k)
Heggernes + H.L.P. (2011):	Bipartite	exp(exp(k))

$\mathcal{G}$ – Contraction: FPT

Heggernes + H.L.L.P. (2011):	Path | Tree	exp(k)
Golovach + K.P.T. (2011):	min-deg >= 14	exp(k)
Heggernes + H.L.P. (2011):	Bipartite	exp(exp(k))
Golovach + H.P. (2012):	Planar	f(k)

$\mathcal{G}$ – Contraction: FPT

Heggernes + H.L.L.P. (2011):	Path | Tree	exp(k)
Golovach + K.P.T. (2011):	min-deg >= 14	exp(k)
Heggernes + H.L.P. (2011):	Bipartite	exp(exp(k))
Golovach + H.P. (2012):	Planar	f(k)
Guillemont & Marx (2013):	Bipartite	exp(k^2)

$\mathcal{G}$ – Contraction: FPT

Heggernes + H.L.L.P. (2011):	Path | Tree	exp(k)
Golovach + K.P.T. (2011):	min-deg >= 14	exp(k)
Heggernes + H.L.P. (2011):	Bipartite	exp(exp(k))
Golovach + H.P. (2012):	Planar	f(k)
Guillemont & Marx (2013):	Bipartite	exp(k^2)
Belmonte + G.H.P.(2013):	max-deg <= 2	exp(k log(k))

$\mathcal{G}$ – Contraction: FPT

Heggernes + H.L.L.P. (2011):	Path | Tree	exp(k)
Golovach + K.P.T. (2011):	min-deg >= 14	exp(k)
Heggernes + H.L.P. (2011):	Bipartite	exp(exp(k))
Golovach + H.P. (2012):	Planar	f(k)
Guillemont & Marx (2013):	Bipartite	exp(k^2)
Belmonte + G.H.P.(2013):	max-deg <= 2	exp(k log(k))
Guo & Cai (2015):	Clique	exp(k log k)

$\mathcal{G}$ – Contraction: Kernel

Heggernes + H.L.L.P. (2011):	Path	k^2
	Tree	no-poly
	min-deg >= 14	??
	Bipartite	??
	Planar	??
Belmonte + G.H.P.(2013):	max-deg <= 2	k^2
Guo & Cai (2015):	Clique	no-poly

$\mathcal{G}$ – Contraction: W[i]-Hardness

$\mathcal{G}$ – Contraction: W[i]-Hardness

Golovach + K.P.T.(2011): min-deg >= d W[1] (only k)

$\mathcal{G}$ – Contraction: W[i]-Hardness

Golovach + K.P.T.(2011): min-deg >= d W[1] (only k)

Belmonte + G.H.P.(2013): max-deg <= d W[2] (only k)

$\mathcal{G}$ – Contraction: W[i]-Hardness

Golovach + K.P.T.(2011):	min-deg >= d	W[1] (only k)
Belmonte + G.H.P.(2013):	max-deg <= d	W[2] (only k)
Cai & Gua(2013) + Lokshtanov + M.S.(2013):	Chordal	W[2]

$\mathcal{G}$ – Contraction: W[i]-Hardness

Golovach + K.P.T.(2011):	min-deg >= d	W[1] (only k)
Belmonte + G.H.P.(2013):	max-deg <= d	W[2] (only k)
Cai & Gua(2013) + Lokshtanov + M.S.(2013):	Chordal	W[2]
Agrawal + L.S.Z.(2017):	Split	W[1]

Comparing Graph Modification problems.

Comparing Graph Modification problems.

(1) vertex deletion
(2) edge deletion
(3) edge addition
(4) edge contraction

Comparing Graph Modification problems.

(1) vertex deletion
(2) edge deletion
(3) edge addition
(4) edge contraction

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (1) Target Graph Class: {P_4}

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (1) Target Graph Class: $\{P_4\}$

Q1 : Delete vertices in G to get P_4.

Q2 : Contract edges in G to get P_4.

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (1) Target Graph Class: {P_4}

Q1 : Delete vertices in G to get P_4.

(Find induced P_4) Polynomial time

Q2 : Contract edges in G to get P_4.

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (1) Target Graph Class: {P_4}

Q1 : Delete vertices in G to get P_4.

(Find induced P_4) Polynomial time

Q2 : Contract edges in G to get P_4.

NP-Hard

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (2)

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (2) Target Graph Class: Chordal Graphs

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (2) Target Graph Class: Chordal Graphs

Q1 : Add at most k edges in G to get a chordal graph.

Q2 : Contract at most k edges in G to get a chordal graph.

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (2) Target Graph Class: Chordal Graphs

Q1 : Add at most k edges in G to get a chordal graph.

(Minimum Fill-In) FPT

Q2 : Contract at most k edges in G to get a chordal graph.

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (2) Target Graph Class: Chordal Graphs

Q1 : Add at most k edges in G to get a chordal graph.

(Minimum Fill-In) FPT

Q2 : Contract at most k edges in G to get a chordal graph.

W[2]-Hard

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (3)

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (3) Target Graph Class: Acyclic Graphs

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (3) Target Graph Class: Acyclic Graphs

Q1 : Delete at most k vertices to make G acyclic.

Q2 : Contract at most k edges to make G acyclic.

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (3) Target Graph Class: Acyclic Graphs

Q1 : Delete at most k vertices to make G acyclic.

(Feedback Vertex Set) k^2 kernel

Q2 : Contract at most k edges to make G acyclic.

Edge Contraction problems are harder than corresponding vertex/edge deletion/addition problems.

Ex. (3) Target Graph Class: Acyclic Graphs

Q1 : Delete at most k vertices to make G acyclic.

(Feedback Vertex Set) k^2 kernel

Q2 : Contract at most k edges to make G acyclic.

No-poly kernel

Parameterized Complexity and Known Results

Overview Of Our Contributions

Brief Description: Lossy Kernel for Clique Cont.

Parameterized Complexity and Known Results

Overview Of Our Contributions

Brief Description: Lossy Kernel for Clique Cont.

Starting Point: "Contracting Graphs to Paths and Trees" by Heggernes et al. (IPEC 2012, Algorithmica 2014)

	Path	Tree
FPT	2^k	4^k
Kernel	k^2	no-poly

Starting Point: "Contracting Graphs to Paths and Trees" by Heggernes et al. (IPEC 2012, Algorithmica 2014)

	Path	Tree
FPT	2^k	4^k
Kernel	k^2	no-poly

Q1. Are there "tree-like" graph classes $\mathcal{G}$ for which $\mathcal{G}$-Contraction is FPT?

Starting Point: "Contracting Graphs to Paths and Trees" by Heggernes et al. (IPEC 2012, Algorithmica 2014)

	Path	Tree
FPT	2^k	4^k
Kernel	k^2	no-poly

Starting Point: "Contracting Graphs to Paths and Trees" by Heggernes et al. (IPEC 2012, Algorithmica 2014)

	Path		Tree
FPT	2^k		4^k
Kernel	k^2	⟷	no-poly

Starting Point: "Contracting Graphs to Paths and Trees" by Heggernes et al. (IPEC 2012, Algorithmica 2014)

	Path		Tree
FPT	2^k		4^k
Kernel	k^2	⟷	no-poly

Q2. Can we interpolate between these two extremes?

Starting Point: "Contracting Graphs to Paths and Trees" by Heggernes et al. (IPEC 2012, Algorithmica 2014)

	Path	Tree
FPT	2^k	4^k
Kernel	k^2	no-poly

Starting Point: "Contracting Graphs to Paths and Trees" by Heggernes et al. (IPEC 2012, Algorithmica 2014)

	Path	Tree
FPT	2^k	4^k
Kernel	k^2	no-poly

Q3. If we are allowed some "loss in accuracy", then can we get a poly-kernel for Tree Cont.?

Q1. Are there "tree-like" graph classes $\mathcal{G}$ for which $\mathcal{G}$-Contraction is FPT?

Q2. Can we interpolate between two extremes?

Path-Cont. – poly kernel | Tree-Cont. – no poly kernel

Q3. If we are allowed some "loss in accuracy", then can we get a poly-kernel for Tree Cont.?

Q1. Are there "tree-like" graph classes $\mathcal{G}$ for which $\mathcal{G}$-Contraction is FPT?

[R1] Cactus

Q2. Can we interpolate between two extremes?

Path-Cont. – poly kernel | Tree-Cont. – no poly kernel

Q3. If we are allowed some "loss in accuracy", then can we get a poly-kernel for Tree Cont.?

Q1. Are there "tree-like" graph classes $\mathcal{G}$ for which $\mathcal{G}$-Contraction is FPT?

[R1] Cactus

[R2] (Tree + q edges)

Q2. Can we interpolate between two extremes?

Path-Cont. – poly kernel | Tree-Cont. – no poly kernel

Q3. If we are allowed some "loss in accuracy", then can we get a poly-kernel for Tree Cont.?

Q1. Are there "tree-like" graph classes $\mathcal{G}$ for which $\mathcal{G}$-Contraction is FPT?

[R1] Cactus

[R2] (Tree + q edges)

Q2. Can we interpolate between two extremes?

Path-Cont. – poly kernel | Tree-Cont. – no poly kernel

[R3] Bounded Tree Cont. – Admits poly kernel.

Q3. If we are allowed some "loss in accuracy", then can we get a poly-kernel for Tree Cont.?

Q1. Are there "tree-like" graph classes $\mathcal{G}$ for which $\mathcal{G}$-Contraction is FPT?

[R1] Cactus

[R2] (Tree + q edges)

Q2. Can we interpolate between two extremes?

Path-Cont. – poly kernel | Tree-Cont. – no poly kernel

[R3] Bounded Tree Cont. – Admits poly kernel.

Q3. If we are allowed some "loss in accuracy", then can we get a poly-kernel for Tree Cont.?

[R4] Lossy Kernel for Tree-Cont.

[R1] Cactus Contraction is FPT.

[R2] (Tree + q edges) Contraction is FPT.

[R3] Bounded Tree Cont. – Admits poly kernel.

[R4] Lossy Kernel for Tree-Cont.

[R1] Cactus Contraction is FPT.

[R2] (Tree + q edges) Contraction is FPT.

[R3] Bounded Tree Cont. – Admits poly kernel.

[R4] Lossy Kernel for Tree-Cont.

[R5] Lossy Kernel for Clique-Cont.

[R1] Cactus Contraction is FPT.

[R2] (Tree + q edges) Contraction is FPT.

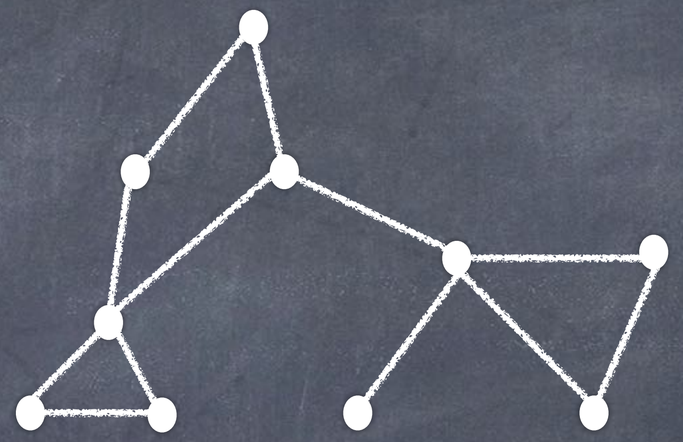
[R3] Bounded Tree Cont. – Admits poly kernel.

[R4] Lossy Kernel for Tree-Cont.

[R5] Lossy Kernel for Clique-Cont.

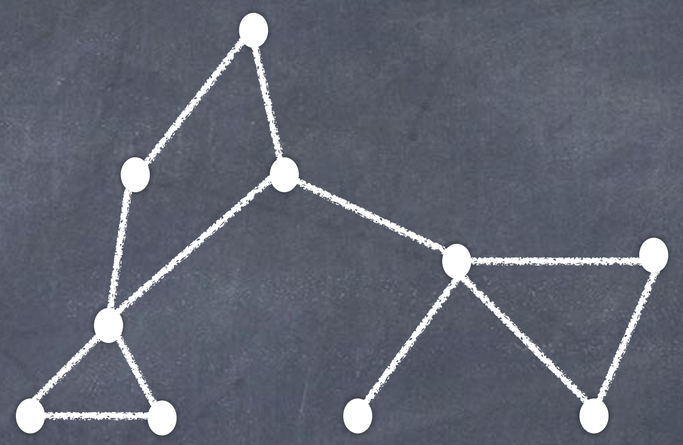
[R6] Exact Exponential Algo for Path-Cont.

[R1] Cactus Contraction is FPT.



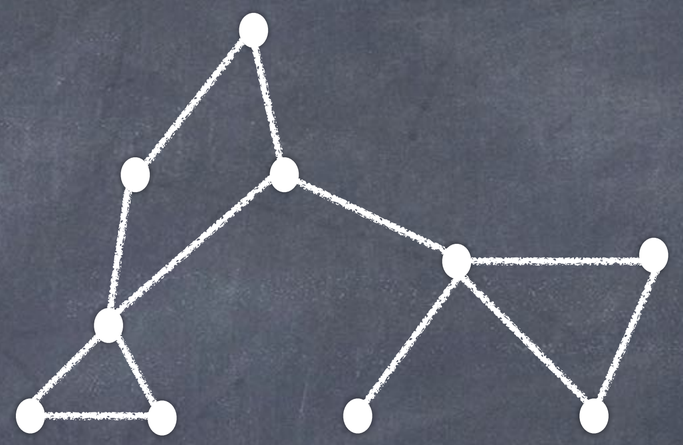
[R1] Cactus Contraction is FPT.

Cactus : A connected graph in which every edge is a part of at most one cycle.



[R1] Cactus Contraction is FPT.

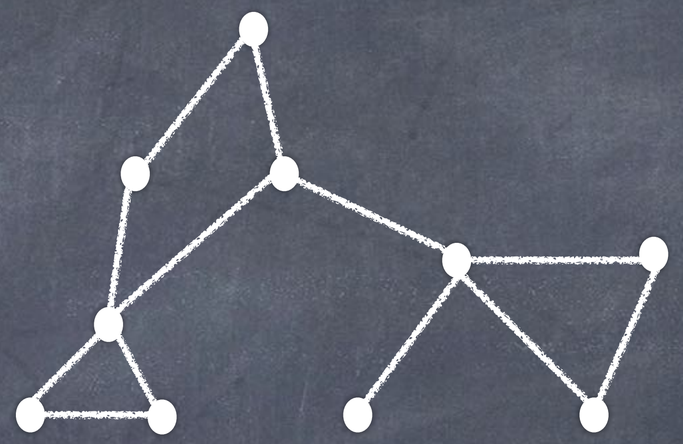
Cactus : A connected graph in which every edge is a part of at most one cycle.



Theorem: There exists an FPT algorithm running in time c^k poly(n) that solves Cactus Contraction.

[R1] Cactus Contraction is FPT.

Cactus : A connected graph in which every edge is a part of at most one cycle.



Theorem: There exists an FPT algorithm running in time c^k poly(n) that solves Cactus Contraction.

– Krithika, Mishra, and T. (COCOON 2018)

[R1] Cactus Contraction is FPT.

[R2] (Tree + q edges) Contraction is FPT.

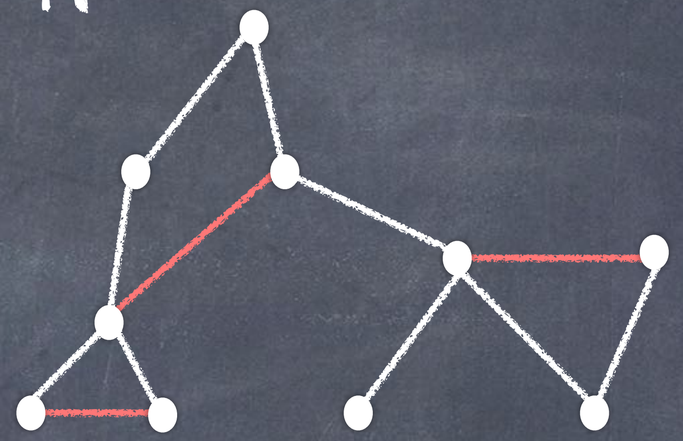
[R3] Bounded Tree Contraction – Admits poly kernel.

[R4] Lossy Kernel for Tree-Cont.

[R5] Lossy Kernel for Clique-Cont.

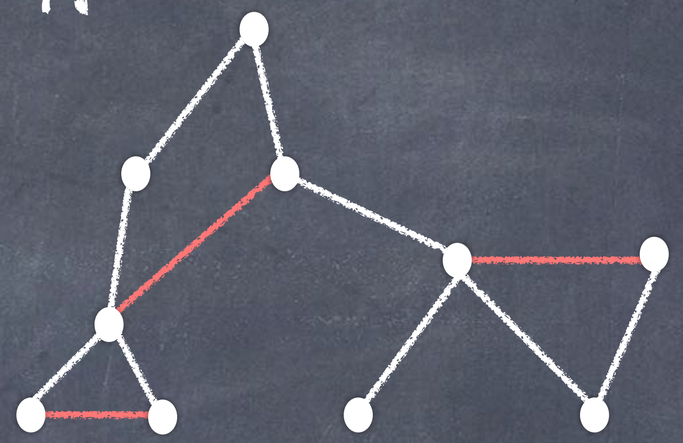
[R6] Exact Exponential Algo for Path-Cont.

[R2] (Tree + q edges) Contraction is FPT.

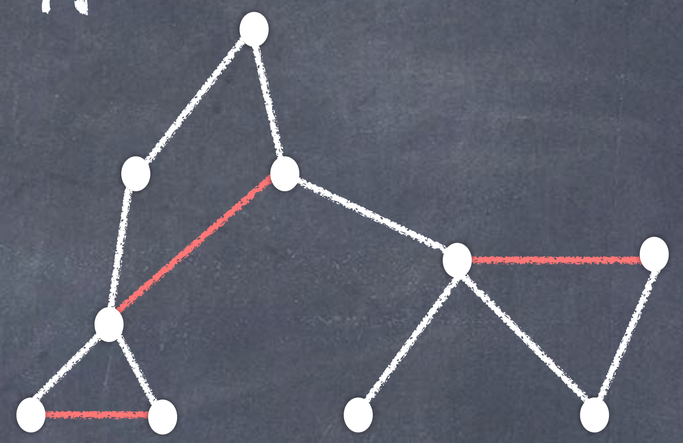


[R2] (Tree + q edges) Contraction is FPT.

Collection of graphs that can be obtained from a tree by adding at most q edges.



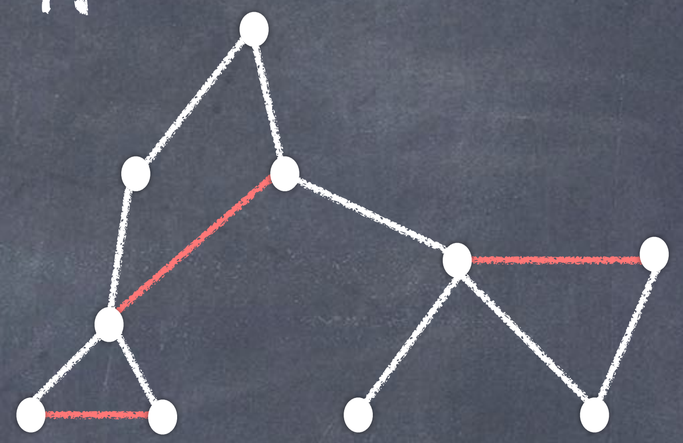
[R2] (Tree + q edges) Contraction is FPT.



Collection of graphs that can be obtained from a tree by adding at most q edges.

Tr_q = {H | Feedback edge set of (H) <= q}

[R2] (Tree + q edges) Contraction is FPT.

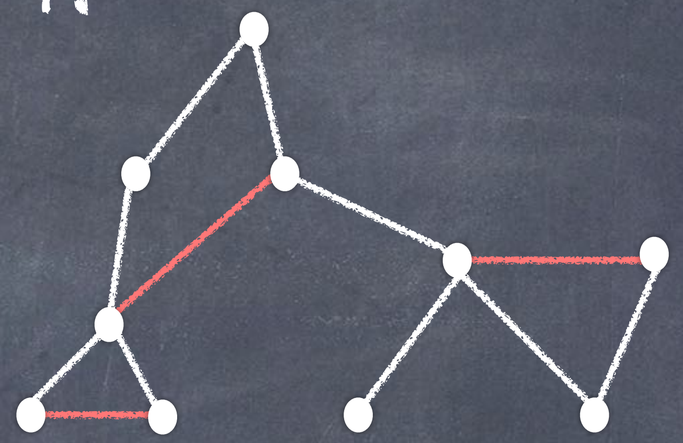


Collection of graphs that can be obtained from a tree by adding at most q edges.

Tr_q = {H | Feedback edge set of (H) <= q}

Theorem: There exists an FPT algorithm running in time (\sqrt(q) + 2)^(k + q) poly(n) that solves Tr_q-Contraction.

[R2] (Tree + q edges) Contraction is FPT.



Collection of graphs that can be obtained from a tree by adding at most q edges.

Tr_q = {H | Feedback edge set of (H) <= q}

Theorem: There exists an FPT algorithm running in time (\sqrt(q) + 2)^(k + q) poly(n) that solves Tr_q-Contraction.

— Agrawal, Saurabh, and T. (IPEC 2017)

[R1] Cactus Contraction is FPT.

[R2] (Tree + q edges) Contraction is FPT.

[R3] Bounded Tree Contraction – Admits poly kernel.

[R4] Lossy Kernel for Tree-Cont.

[R5] Lossy Kernel for Clique-Cont.

[R6] Exact Exponential Algo for Path-Cont.

[R3] Bounded Tree Contraction – Admits poly kernel.

[R3] Bounded Tree Contraction – Admits poly kernel.

Bounded-Tree(l) : {T | T is a tree with at most l leaves}

[R3] Bounded Tree Contraction – Admits poly kernel.

Bounded-Tree(l) : {T | T is a tree with at most l leaves}

Collection of paths = Bounded-Tree(2)

[R3] Bounded Tree Contraction – Admits poly kernel.

Bounded-Tree(l) : {T | T is a tree with at most l leaves}

Collection of paths = Bounded-Tree(2)

Collection of stars = Bounded-Tree(n - 1)

[R3] Bounded Tree Contraction – Admits poly kernel.

Bounded-Tree(l) : {T | T is a tree with at most l leaves}

Collection of paths = Bounded-Tree(2)

Collection of stars = Bounded-Tree(n - 1)

Theorem: Bounded-Tree Contraction admits a kernel of size (k^2 + kl).

[R3] Bounded Tree Contraction – Admits poly kernel.

Bounded-Tree(l) : {T | T is a tree with at most l leaves}

Collection of paths = Bounded-Tree(2)

Collection of stars = Bounded-Tree(n - 1)

Theorem: Bounded-Tree Contraction admits a kernel of size (k^2 + kl).

Path Contraction – k^2 kernel

Tree Contraction – no poly kernel

[R3] Bounded Tree Contraction – Admits poly kernel.

Theorem: Bounded-Tree Contraction admits a kernel of size (k^2 + kl).

Theorem: Bounded-Tree Contraction does not admit a kernel of size $(k^2 + kl)^{1-\epsilon}$ for any $\epsilon > 0$ unless NP $\subseteq$ coNP/poly.

– Agrawal, Kanesh, Saurabh, and T. (CIAC 2017)

[R3] Bounded Tree Contraction – Admits poly kernel.

Theorem: Bounded-Tree Contraction admits a kernel of size (k^2 + kl).

Theorem: Bounded-Tree Contraction does not admit a kernel of size $(k^2 + kl)^{1-\epsilon}$ for any $\epsilon > 0$ unless NP $\subseteq$ coNP/poly.

Similar results for

– Agrawal, Kanesh, Saurabh, and T. (CIAC 2017)

[R3] Bounded Tree Contraction – Admits poly kernel.

Theorem: Bounded-Tree Contraction admits a kernel of size (k^2 + kl).

Theorem: Bounded-Tree Contraction does not admit a kernel of size $(k^2 + kl)^{1-\epsilon}$ for any $\epsilon > 0$ unless NP $\subseteq$ coNP/poly.

Similar results for

- Bounded Cactus Contraction

- Agrawal, Kanesh, Saurabh, and T. (CIAC 2017)

[R3] Bounded Tree Contraction – Admits poly kernel.

Theorem: Bounded-Tree Contraction admits a kernel of size (k^2 + kl).

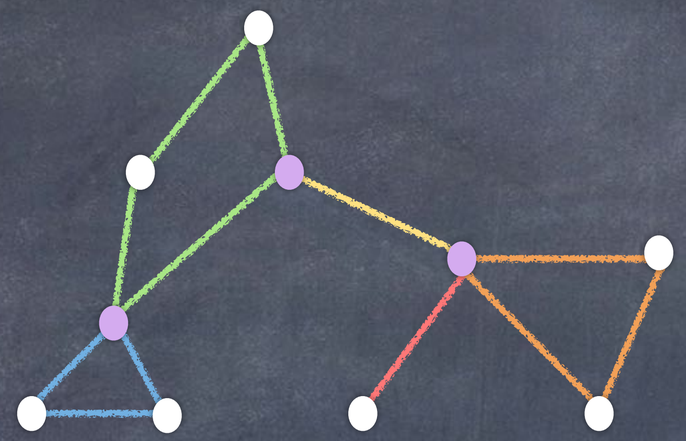
Theorem: Bounded-Tree Contraction does not admit a kernel of size $(k^2 + kl)^{1-\epsilon}$ for any $\epsilon > 0$ unless NP $\subseteq$ coNP/poly.

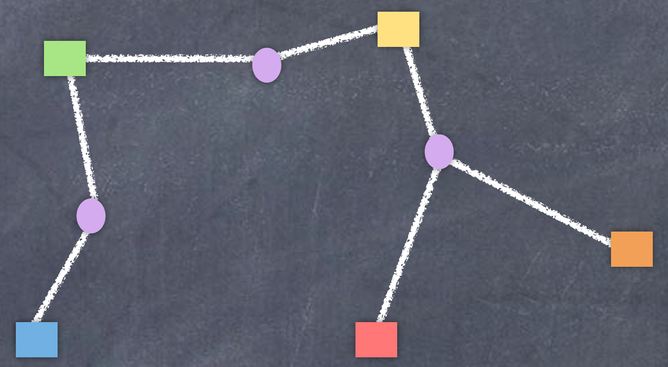
Similar results for

- Bounded Cactus Contraction
- Bounded Out-tree Contraction

– Agrawal, Kanesh, Saurabh, and T. (CIAC 2017)

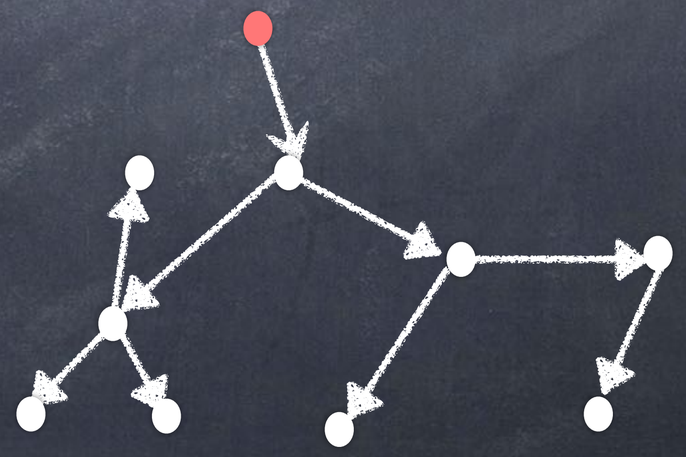
– Bounded Cactus Contraction





– Bounded Out-tree Contraction

rooted-tree, directed, arc
away from root



[R1] Cactus Contraction is FPT.

[R2] (Tree + q edges) Contraction is FPT.

[R3] Bounded Tree Contraction – Admits poly kernel.

[R4] Lossy Kernel for Tree-Cont.

[R5] Lossy Kernel for Clique-Cont.

[R6] Exact Exponential Algo for Path-Cont.

[R4] Lossy Kernel for Tree-Cont.

(Classical) Kernels - decision version of the problem.

[R4] Lossy Kernel for Tree-Cont.

(Classical) Kernels - decision version of the problem.

An approx. sol for the output instance may not provide any insight about an approx. sol for the input instance.

[R4] Lossy Kernel for Tree-Cont.

(Classical) Kernels - decision version of the problem.

An approx. sol for the output instance may not provide any insight about an approx. sol for the input instance.

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2018)

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2018)

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2018)

For $\alpha > 1$, an α-lossy kernel for optimisation version of the problem.

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2018)

For $\alpha > 1$, an α-lossy kernel for optimisation version of the problem.

- Two poly time algo.

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2018)

For $\alpha > 1$, an α-lossy kernel for optimisation version of the problem.

- Two poly time algo.

- Reduction Algo: input (G, k) produces output (G', k')

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2018)

For $\alpha > 1$, an α-lossy kernel for optimisation version of the problem.

- Two poly time algo.

- Reduction Algo: input (G, k) produces output (G', k')

- Sol-lifting Algo: Given c-factor solution S' for (G', k') produces an $(\alpha\, c)$-factor solution S for (G, k).

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2018)

For $\alpha > 1$, an α-lossy kernel for optimisation version of the problem.

- Two poly time algo.

- Reduction Algo: input (G, k) produces output (G', k')

- Sol-lifting Algo: Given c-factor solution S' for (G', k') produces an $(\alpha\, c)$-factor solution S for (G, k).

- Allowed to fail if S' is bad.

α lossy kernel = reduction algo + Sol-lifting algo

α lossy kernel = reduction algo + Sol-lifting algo

(G, k)

Reduction algo

(G', k')

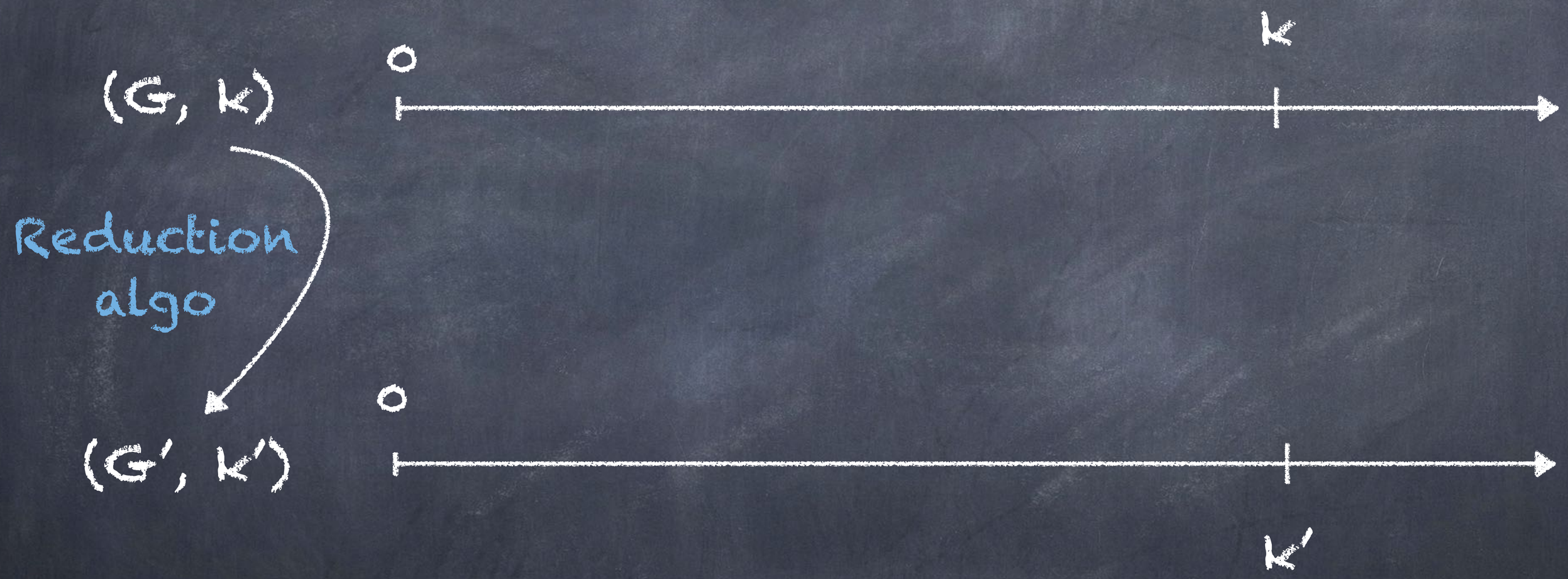
a lossy kernel = reduction algo + Sol-lifting algo

(G, k)

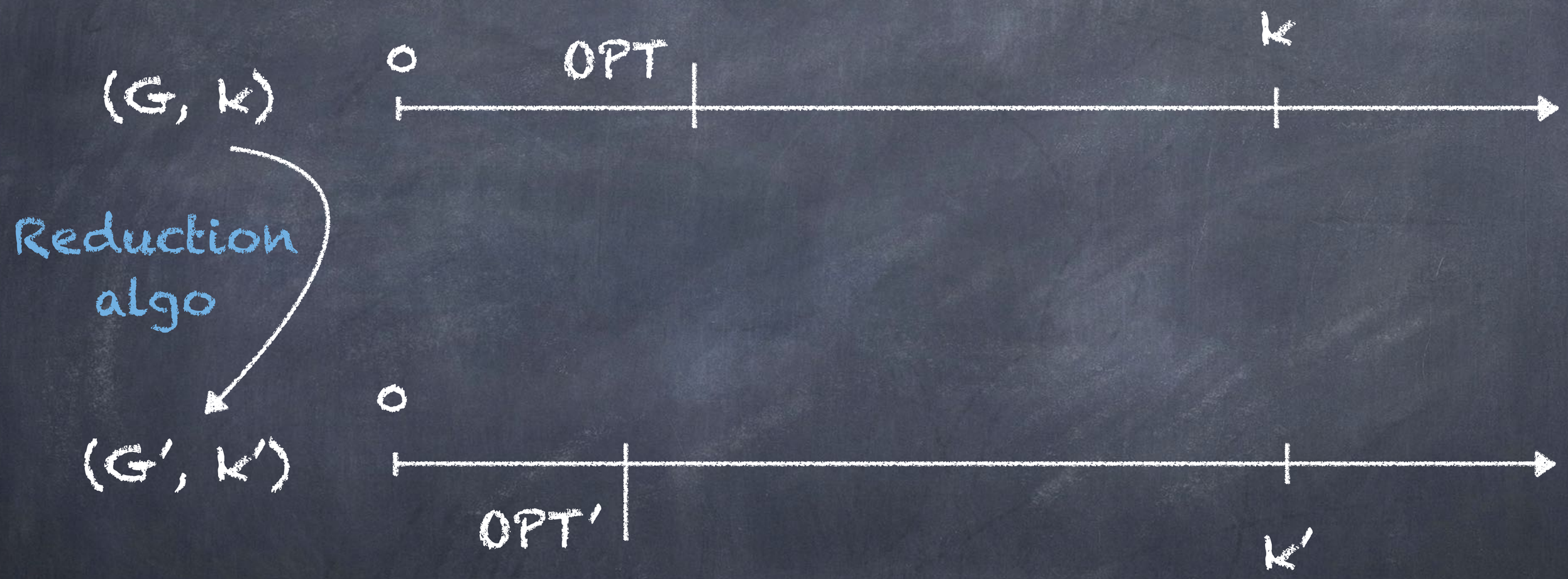
Reduction algo

(G', k')

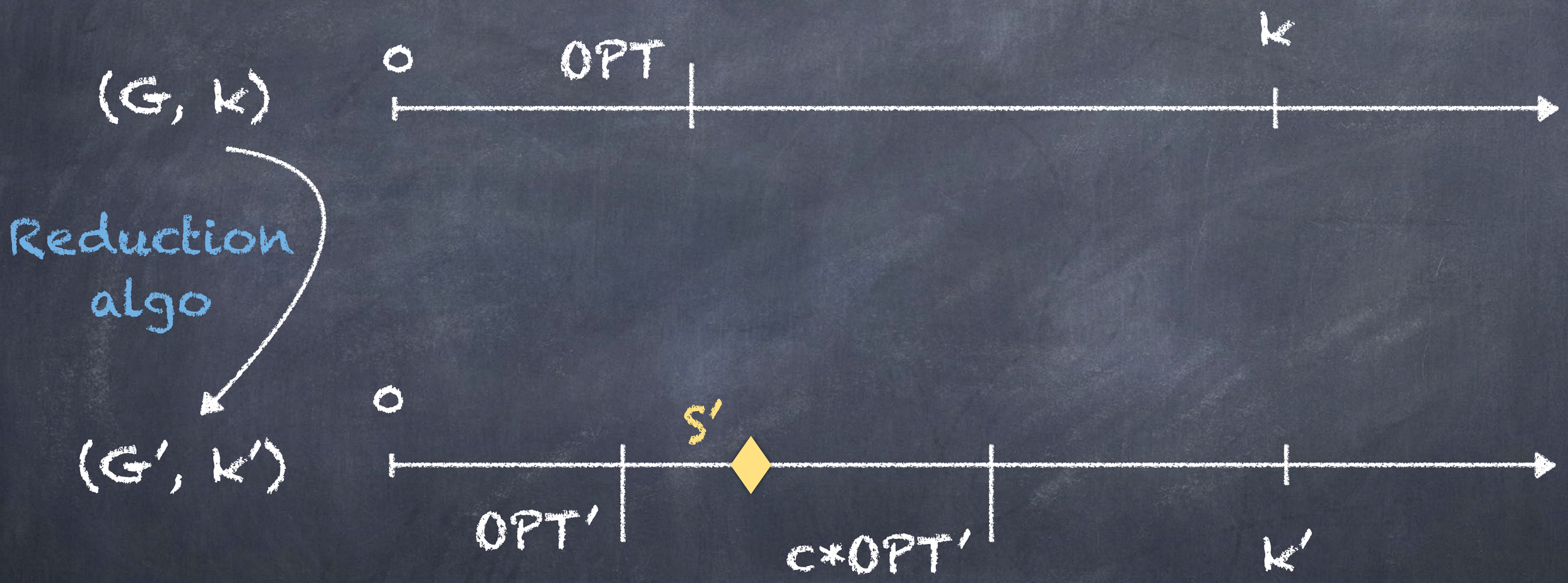
a lossy kernel = reduction algo + Sol-lifting algo



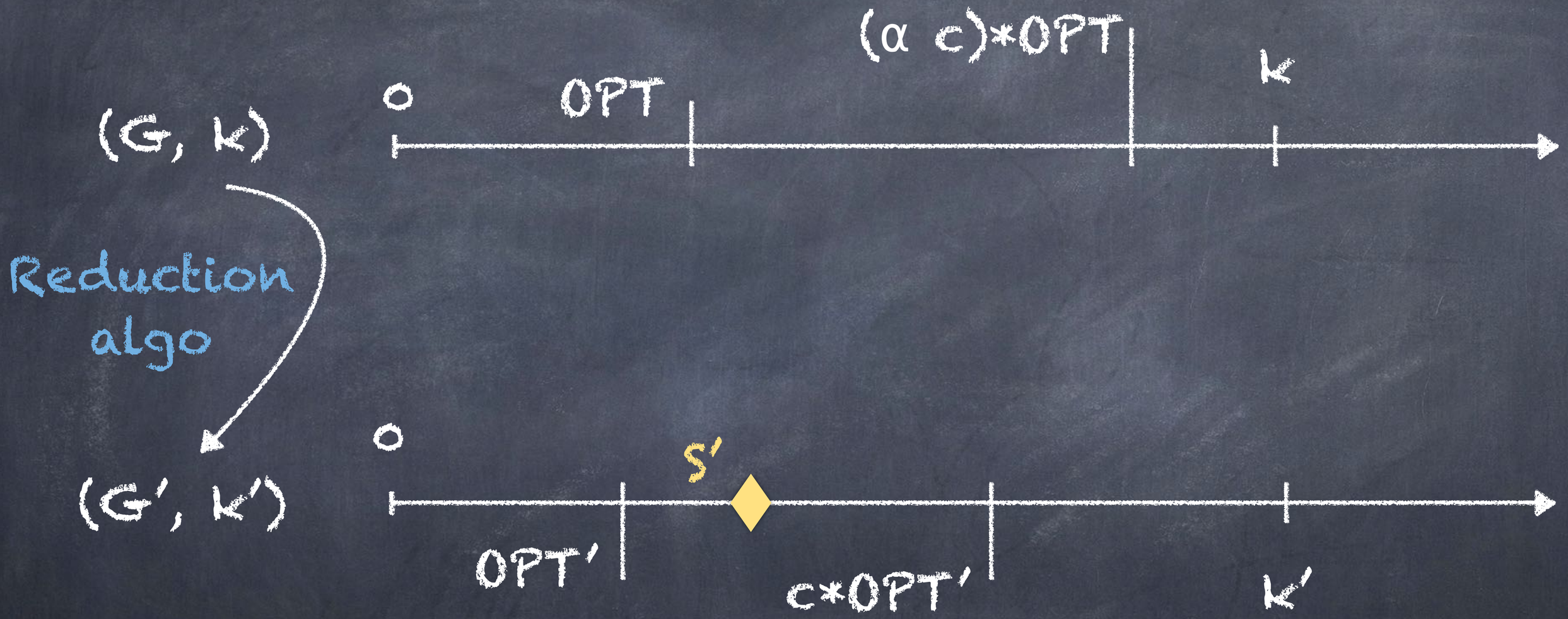
α lossy kernel = reduction algo + Sol-lifting algo



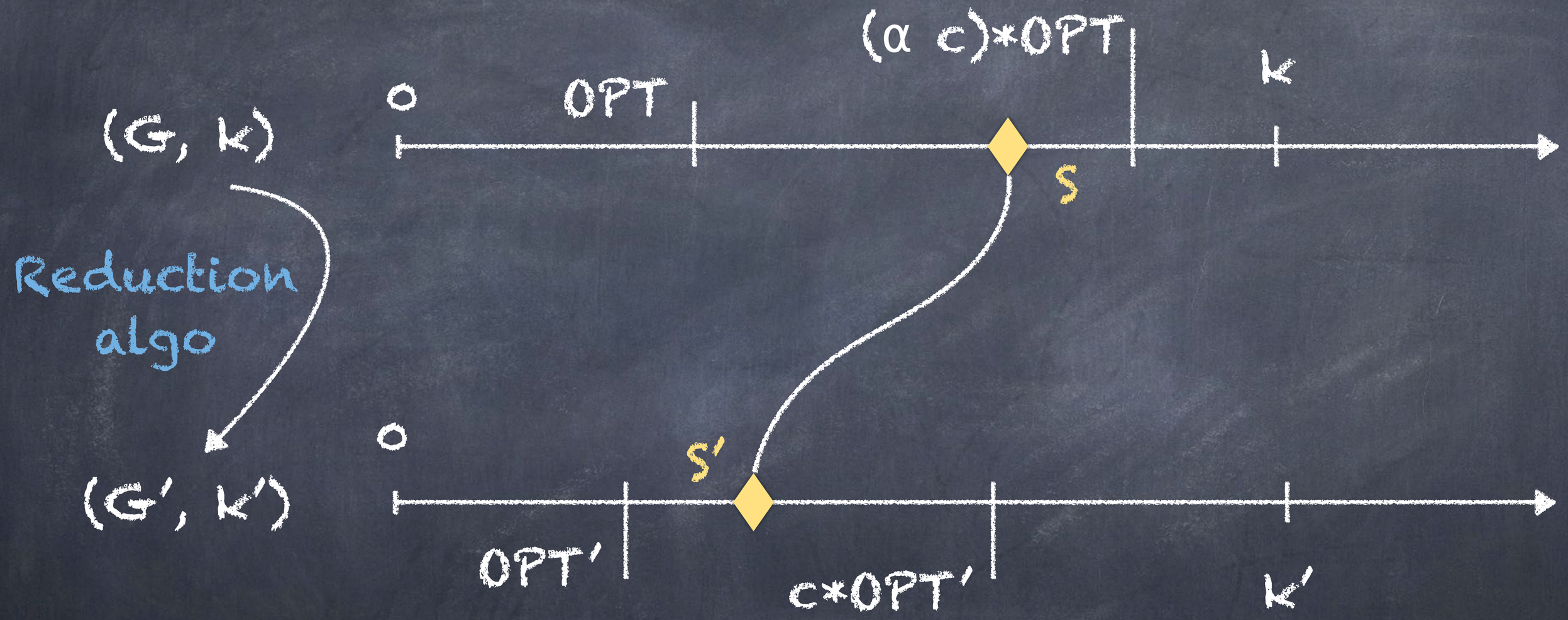
α lossy kernel = reduction algo + Sol-lifting algo



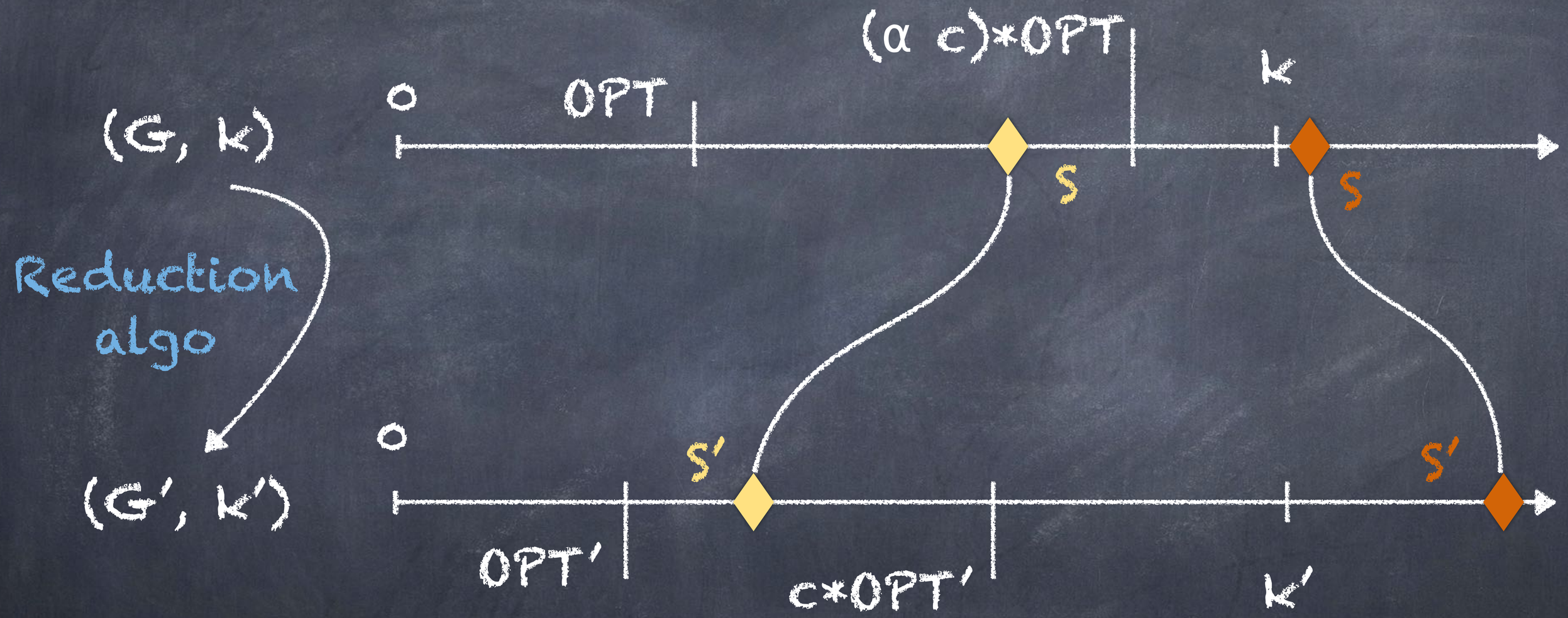
α lossy kernel = reduction algo + Sol-lifting algo



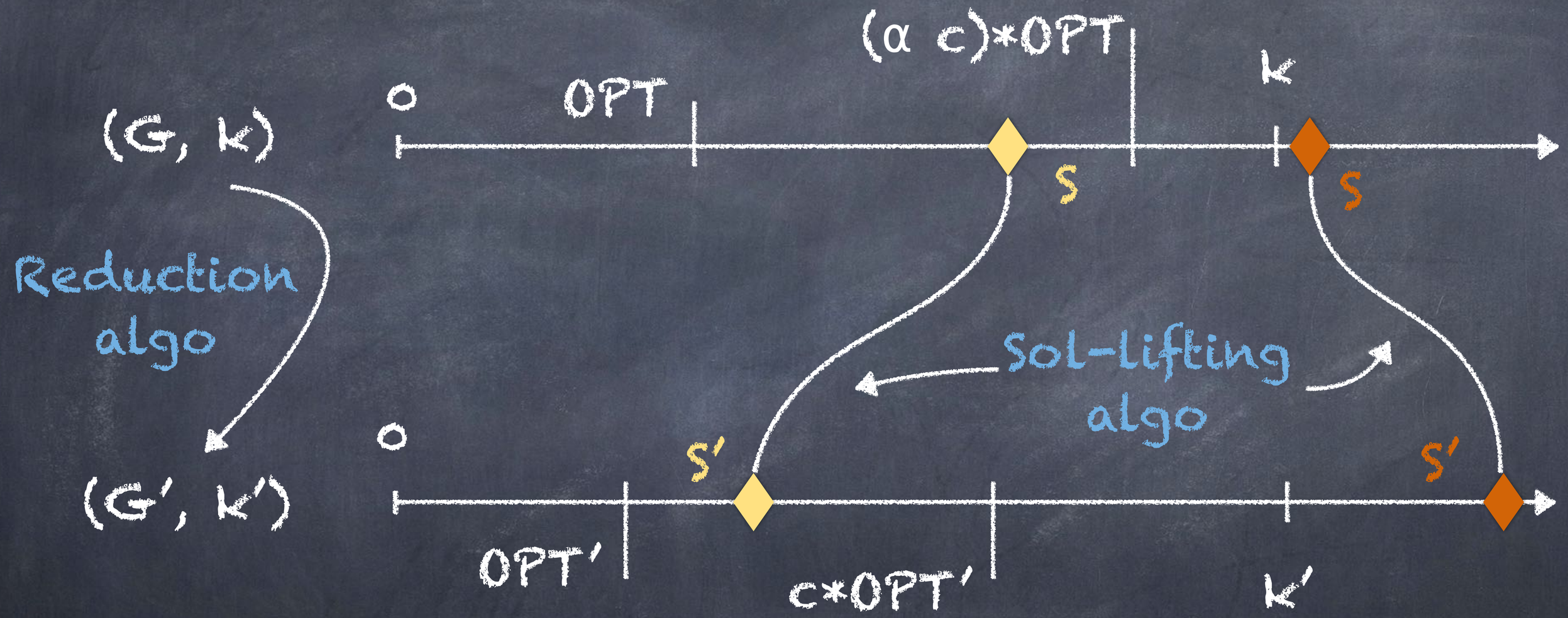
α lossy kernel = reduction algo + Sol-lifting algo



α lossy kernel = reduction algo + Sol-lifting algo



α lossy kernel = reduction algo + Sol-lifting algo



"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2017)

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2017)

α- Lossy Kernels of poly size for

- Connected Vertex Cover
- Disjoint Cycle Packing
- Disjoint Factors

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2017)

α- Lossy Kernels of poly size for

- Connected Vertex Cover
- Disjoint Cycle Packing
- Disjoint Factors

No poly (classical) kernel

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2017)

α- Lossy Kernels of poly size for

- Connected Vertex Cover
- Disjoint Cycle Packing
- Disjoint Factors

No poly (classical) kernel
Poly kernel size – $k^{f(\alpha)}$

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2017)

α- Lossy Kernels of poly size for

- Connected Vertex Cover
- Disjoint Cycle Packing
- Disjoint Factors

No poly (classical) kernel
Poly kernel size – $k^{f(\alpha)}$

No Lossy kernels of poly size for

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2017)

α- Lossy Kernels of poly size for

- Connected Vertex Cover
- Disjoint Cycle Packing
- Disjoint Factors

No poly (classical) kernel
Poly kernel size – $k^{f(\alpha)}$

No Lossy kernels of poly size for

- Long Path
- Set Cover/universe size

"Lossy Kernelization" Lokshtanov+P.R.S.(STOC 2017)

α- Lossy Kernels of poly size for

- Connected Vertex Cover
- Disjoint Cycle Packing
- Disjoint Factors

No poly (classical) kernel
Poly kernel size – $k^{f(\alpha)}$

No Lossy kernels of poly size for

- Long Path
- Set Cover/universe size

unless $NP \subseteq coNP/poly$

[R4] Lossy Kernel for Tree-Cont.

[R4] Lossy Kernel for Tree-Cont.

Heggernes + H.L.L.P. (2011): Tree-Cont. does not admit a poly (classical) kernel.

[R4] Lossy Kernel for Tree-Cont.

Heggernes + H.L.L.P. (2011): Tree-Cont. does not admit a poly (classical) kernel.

Theorem: For every $\alpha > 1$, Tree-Cont. admits an α-lossy kernel of size $k^{f(\alpha)}$.

[R4] Lossy Kernel for Tree-Cont.

Heggernes + H.L.L.P. (2011): Tree-Cont. does not admit a poly (classical) kernel.

Theorem: For every $\alpha > 1$, Tree-Cont. admits an α-lossy kernel of size $k^{f(\alpha)}$.

- Krithika, Mishra, Rai, and T. (FSTTCS 2016)

[R4] Lossy Kernel for Tree-Cont.

Heggernes + H.L.L.P. (2011): Tree-Cont. does not admit a poly (classical) kernel.

Theorem: For every $\alpha > 1$, Tree-Cont. admits an α-lossy kernel of size $k^{f(\alpha)}$.

For Cactus-Cont. and Out-Tree-Cont.

- Krithika, Mishra, Rai, and T. (FSTTCS 2016)

[R4] Lossy Kernel for Tree-Cont.

Heggernes + H.L.L.P. (2011): Tree-Cont. does not admit a poly (classical) kernel.

Theorem: For every $\alpha > 1$, Tree-Cont. admits an α-lossy kernel of size $k^{f(\alpha)}$.

For Cactus-Cont. and Out-Tree-Cont.

No poly (classical) kernel

- Krithika, Mishra, Rai, and T. (FSTTCS 2016)

[R4] Lossy Kernel for Tree-Cont.

Heggernes + H.L.L.P. (2011): Tree-Cont. does not admit a poly (classical) kernel.

Theorem: For every $\alpha > 1$, Tree-Cont. admits an α-lossy kernel of size $k^{f(\alpha)}$.

For Cactus-Cont. and Out-Tree-Cont.

No poly (classical) kernel

Lossy kernel of poly-size

- Krithika, Mishra, Rai, and T. (FSTTCS 2016)

[R1] Cactus Contraction is FPT.

[R2] (Tree + q edges) Contraction is FPT.

[R3] Bounded Tree Contraction – Admits poly kernel.

[R4] Lossy Kernel for Tree-Cont.

[R5] Lossy Kernel for Clique-Cont.

[R6] Exact Exponential Algo for Path-Cont.

[RS] Lossy Kernel for Clique-Cont.

Guo & Cai (2015): Clique-Contraction does not admit a poly (classical) kernel.

[RS] Lossy Kernel for Clique-Cont.

Guo & Cai (2015): Clique-Contraction does not admit a poly (classical) kernel.

Theorem: For every $\alpha > 1$, Clique-Contraction admits an α-lossy kernel of size $k^{f(\alpha)}$.

[RS] Lossy Kernel for Clique-Cont.

Guo & Cai (2015): Clique-Contraction does not admit a poly (classical) kernel.

Theorem: For every $\alpha > 1$, Clique-Contraction admits an α-lossy kernel of size $k^{f(\alpha)}$.

- Gunda, Jain, Lokshtanov, Saurabh and T.

[RS] Lossy Kernel for Clique-Cont.

Theorem: For every $\alpha > 1$, Clique-Contraction admits an α-lossy kernel of size $k^{f(\alpha)}$.

[RS] Lossy Kernel for Clique-Cont.

Theorem: For every $\alpha > 1$, Clique-Contraction admits an α-lossy kernel of size $k^{f(\alpha)}$.

Clique = Graphs of diameter 1

[RS] Lossy Kernel for Clique-Cont.

Theorem: For every $\alpha > 1$, Clique-Contraction admits an α-lossy kernel of size $k^{f(\alpha)}$.

Clique = Graphs of diameter 1

s-Club = Graphs of diameter s

[RS] Lossy Kernel for Clique-Cont.

Theorem: For every $\alpha > 1$, Clique-Contraction admits an α-lossy kernel of size $k^{f(\alpha)}$.

Clique = Graphs of diameter 1

s-Club = Graphs of diameter s

Theorem: For every $\alpha > 1$ and $s > 1$, s-Club-Contraction does not admit an α-lossy kernel of poly size unless $NP \subseteq coNP/poly$

[RS] Lossy Kernel for Clique-Cont.

Theorem: For every $\alpha > 1$, Clique-Contraction admits an α-lossy kernel of size $k^{f(\alpha)}$.

Clique = Graphs of diameter 1

s-Club = Graphs of diameter s

Theorem: For every $\alpha > 1$ and $s > 1$, s-Club-Contraction does not admit an α-lossy kernel of poly size unless $NP \subseteq coNP/poly$

- Saurabh and T.

[R1] Cactus Contraction is FPT.

[R2] (Tree + q edges) Contraction is FPT.

[R3] Bounded Tree Contraction – Admits poly kernel.

[R4] Lossy Kernel for Tree-Cont.

[R5] Lossy Kernel for Clique-Cont.

[R6] Exact Exponential Algo for Path-Cont.

[R6] Exact Exponential Algo for Path-Cont.

[R6] Exact Exponential Algo for Path-Cont.

Edge contraction as Graph partitioning problem

[R6] Exact Exponential Algo for Path-Cont.

Edge contraction as Graph partitioning problem

(1) G can be contracted to H

[R6] Exact Exponential Algo for Path-Cont.

Edge contraction as Graph partitioning problem

(1) G can be contracted to H

(2) V(G) can be partitioned into |V(H)| many parts s.t.

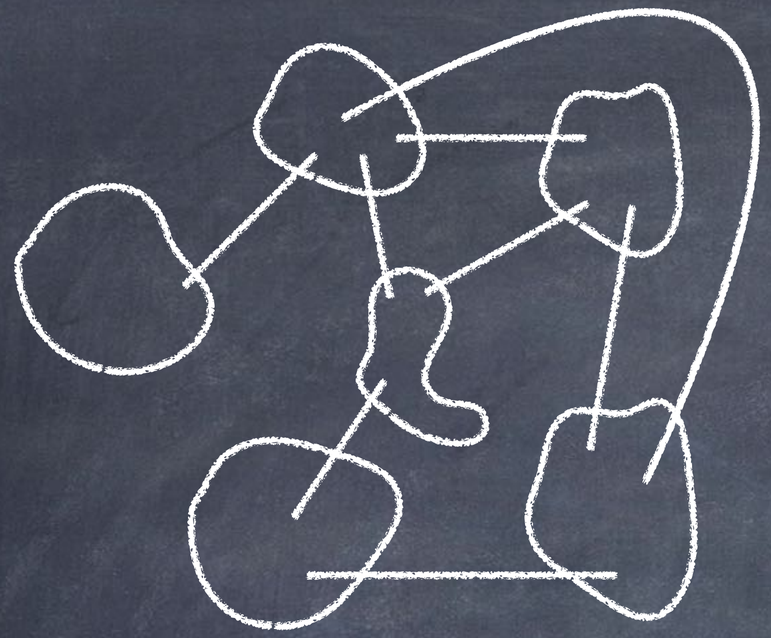
[R6] Exact Exponential Algo for Path-Cont.

Edge contraction as Graph partitioning problem

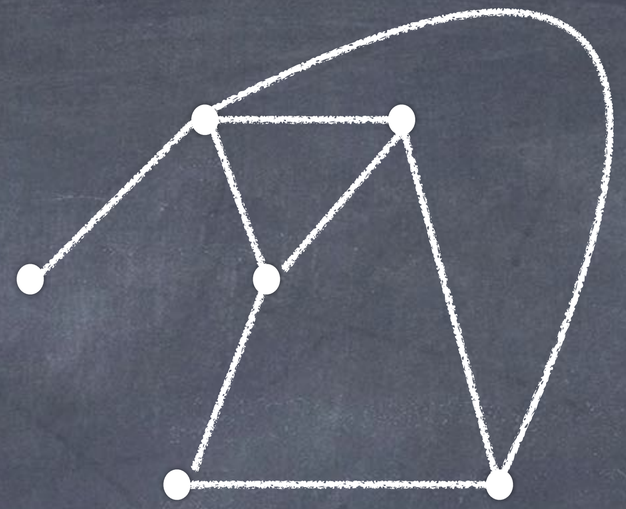
(1) G can be contracted to H

(2) V(G) can be partitioned into |V(H)| many parts s.t.

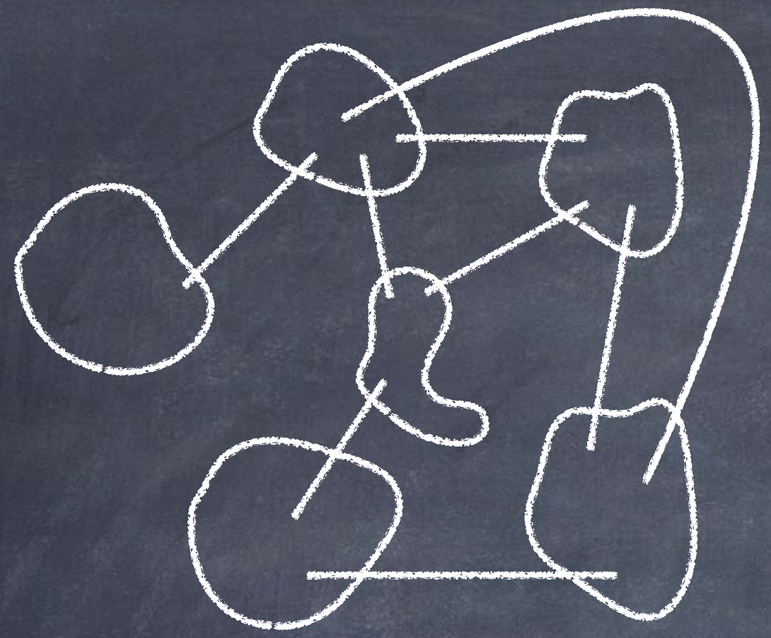
a. Each part is connected
b. Each part is mapped to some vertex is H s.t. V1, V2 adjacent iff h1, h2 are adjacent.



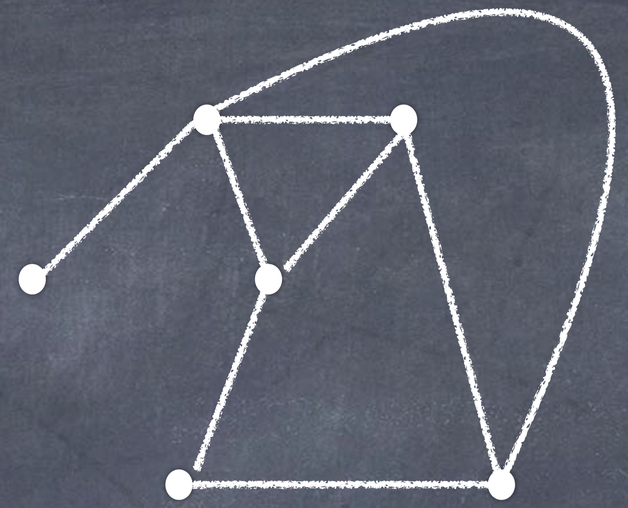
G



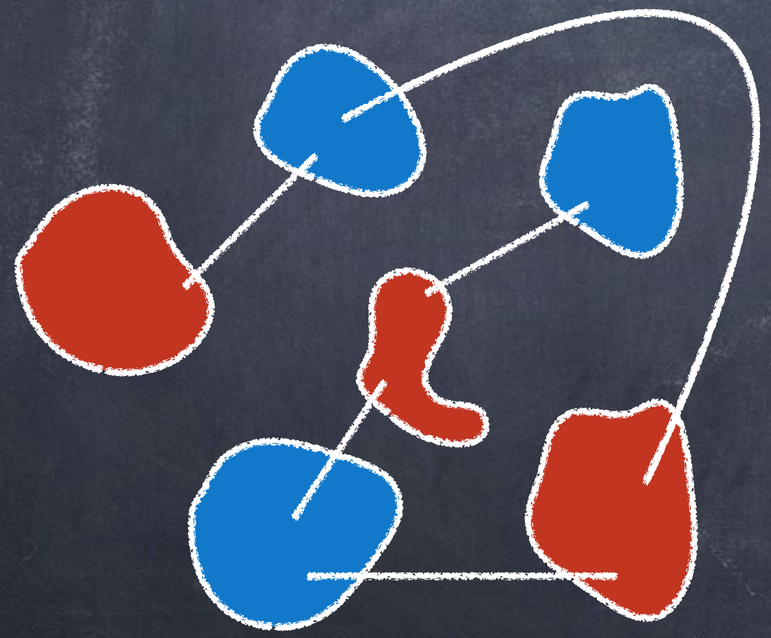
H



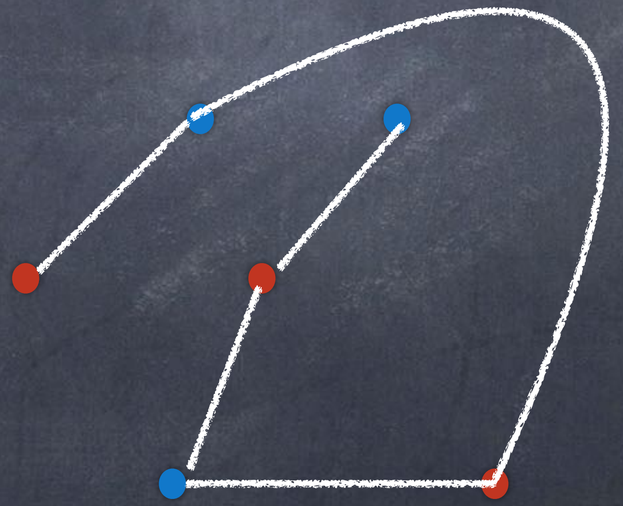
G



H



G



H = Path

For Path-Contraction

For Path-Contraction

- instead of checking all partitions, it is sufficient to check all 2-coloring of V(G).

2^n algo for Path-Contraction

For Path-Contraction

- instead of checking all partitions, it is sufficient to check all 2-coloring of V(G).

2^n algo for Path-Contraction

- Hof + P.W. (2009, TCS): 1.579^n algo for Path-Contraction when input is P_6-free.

For Path-Contraction

- instead of checking all partitions, it is sufficient to check all 2-coloring of V(G).

2^n algo for Path-Contraction

- Hof + P.W. (2009, TCS): 1.579^n algo for Path-Contraction when input is P_6-free.

- Telle + Villanger (2013, WG): 1.78^n algo for {P_4}-Contraction.

For Path-Contraction

- instead of checking all partitions, it is sufficient to check all 2-coloring of V(G).

2^n algo for Path-Contraction

- Hof + P.W. (2009, TCS): 1.579^n algo for Path-Contraction when input is P_6-free.

Can we break 2^n barrier when input is any graph?

- Telle + Villanger (2013, WG): 1.78^n algo for {P_4}-Contraction.

[R6] Exact Exponential Algo for Path-Cont.

- Hof + P.W. (2009, TCS): 1.579^n algo for Path-Contraction when input is P_6-free.

Can we break 2^n barrier when input is any graph?

- Telle + Villanger (2013, WG): 1.78^n algo for {P_4}-Contraction.

[R6] Exact Exponential Algo for Path-Cont.

- Hof + P.W. (2009, TCS): 1.579^n algo for Path-Contraction when input is P_6-free.

Can we break 2^n barrier when input is any graph?

- Telle + Villanger (2013, WG): 1.78^n algo for {P_4}-Contraction.

Theorem: 1.89^n algo for {P_5}-Contraction.

[R6] Exact Exponential Algo for Path-Cont.

- Hof + P.W. (2009, TCS): 1.579^n algo for Path-Contraction when input is P_6-free.

Can we break 2^n barrier when input is any graph?

Theorem: 1.9998^n algorithm for Path-Contraction.

- Telle + Villanger (2013, WG): 1.78^n algo for {P_4}-Contraction.

Theorem: 1.89^n algo for {P_5}-Contraction.

[R6] Exact Exponential Algo for Path-Cont.

- Hof + P.W. (2009, TCS): 1.579^n algo for Path-Contraction when input is P_6-free.

Can we break 2^n barrier when input is any graph?

Theorem: 1.9998^n algorithm for Path-Contraction.

- Telle + Villanger (2013, WG): 1.78^n algo for {P_4}-Contraction.

Theorem: 1.89^n algo for {P_5}-Contraction.

-Agrawal, Fomin, Lokshtanov, Saurabh and T. (ICALP 2019)

[R1] Cactus Contraction is FPT.

[R2] (Tree + q edges) Contraction is FPT.

[R3] Bounded Tree Contraction – Admits poly kernel.

[R4] Lossy Kernel for Tree-Cont.

[R5] Lossy Kernel for Clique-Cont.

[R6] Exact Exponential Algo for Path-Cont.

Parameterized Complexity and Known Results

Overview Of Our Contributions

Brief Description: Lossy Kernel for Clique Cont.

Parameterized Complexity and Known Results

Overview Of Our Contributions

Brief Description: Lossy Kernel for Clique Cont.

[RS] Lossy Kernel for Clique-Cont.

Input : Graph G, integer k **Parameter :** k

Output : Output a set F of edges of minimum cardinality s.t. G/F is a clique.

[RS] Lossy Kernel for Clique-Cont.

Input : Graph G, integer k **Parameter :** k

Output : Output a set F of edges of minimum cardinality s.t. G/F is a clique.

Every solution of size larger than k is equally bad.

[RS] Lossy Kernel for Clique-Cont.

Input : Graph G, integer k **Parameter :** k

Output : Output a set F of edges of minimum cardinality s.t. G/F is a clique.

Every solution of size larger than k is equally bad.

G is connected and has at least $(k + 3)$ vertices

[RS] Lossy Kernel for Clique-Cont.

Input : Graph G, integer k **Parameter :** k

Output : Output a set F of edges of minimum cardinality s.t. G/F is a clique.

Every solution of size larger than k is equally bad.

G is connected and has at least $(k + 3)$ vertices

Any spanning tree is a trivial solution

Observation : If G can be converted into a clique by k-edge contraction then it can also be converted into a clique by 2k-vertex deletion.

Observation : If G can be converted into a clique by k-edge contraction then it can also be converted into a clique by 2k-vertex deletion.

Reduction Rule 1 : Given (G, k) find minimum sized X such that G-X is a clique. If $|X| > 4k$ then return no instance.

Observation : If G can be converted into a clique by k-edge contraction then it can also be converted into a clique by 2k-vertex deletion.

Reduction Rule 1 : Given (G, k) find minimum sized X such that G-X is a clique. If $|X| > 4k$ then return no instance.

- 2-factor approx algo for vertex cover in compliment graph

Reduction Rule 1 : Given (G, k) find minimum sized X such that $G-X$ is a clique. If $|X| > 4k$ then return no instance.

Solution lifting algorithm 1:

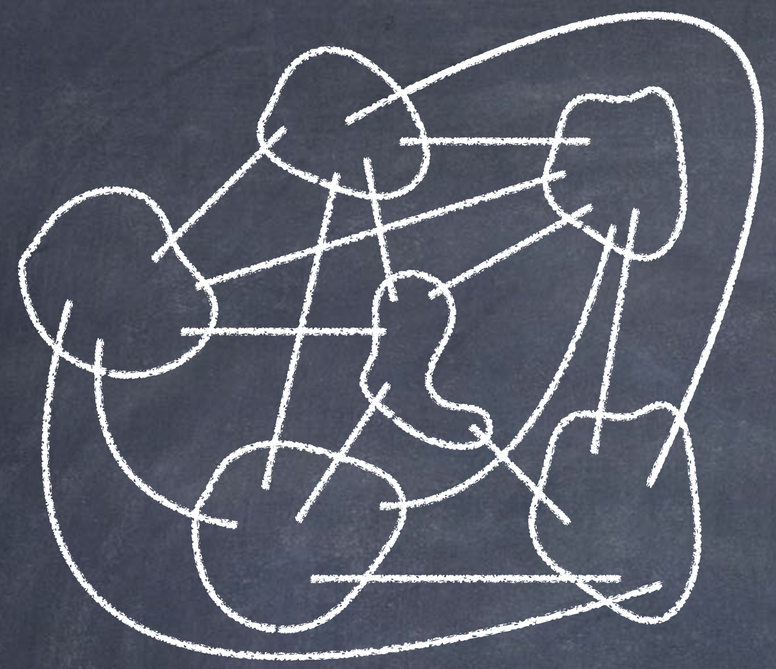
- If un-changed, return same solution.
- If no instance then return a spanning tree.

Reduction Rule 1 : Given (G, k) find minimum sized X such that $G-X$ is a clique. If $|X| > 4k$ then return no instance.

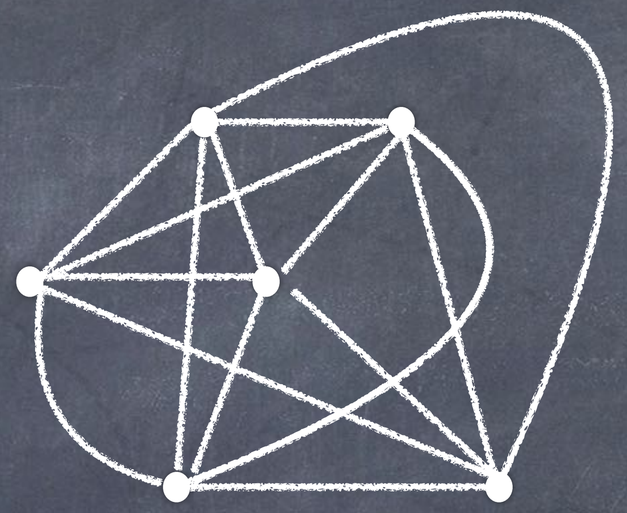
Solution lifting algorithm 1:

- If un-changed, return same solution.
- If no instance then return a spanning tree.

Lemma : RR-1 is 1-safe.



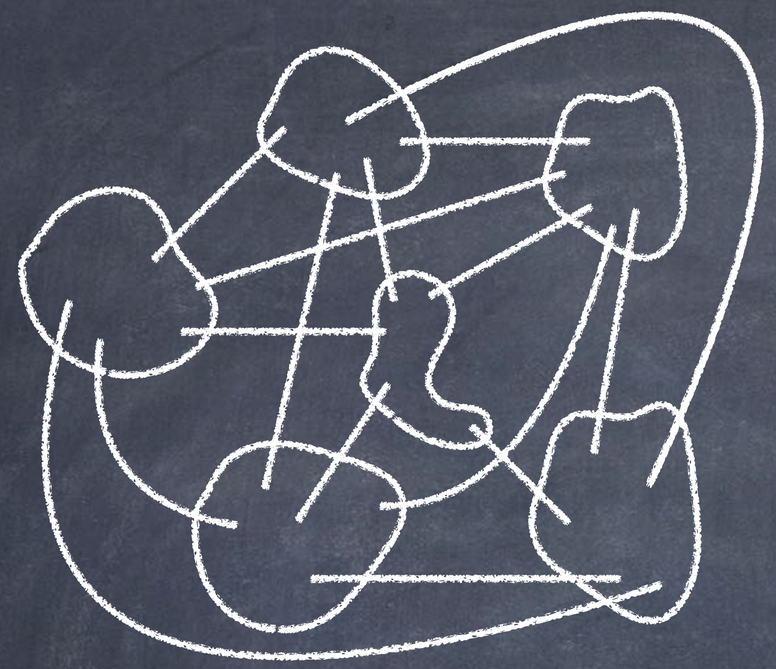
G



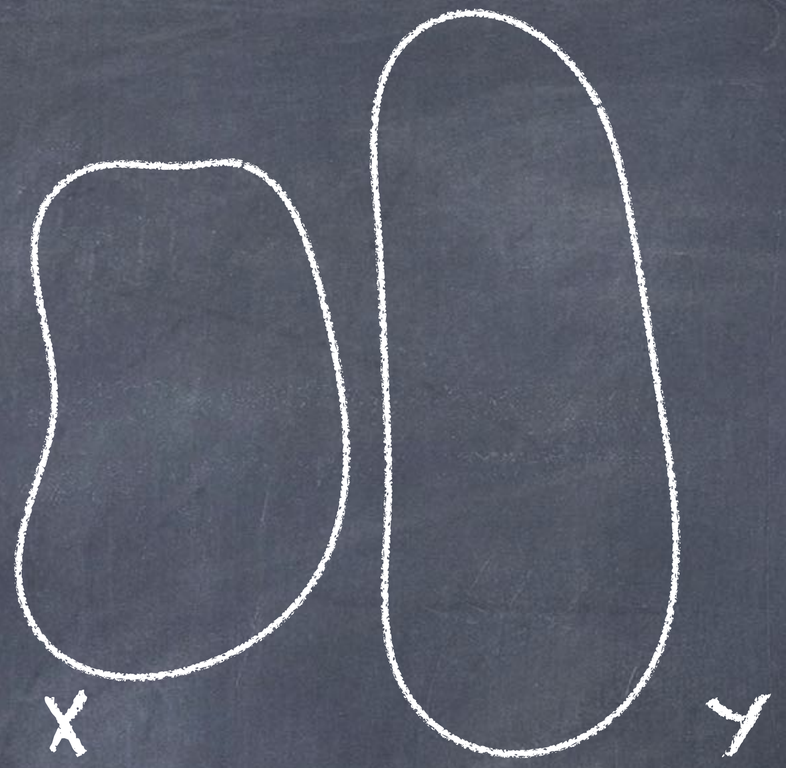
H

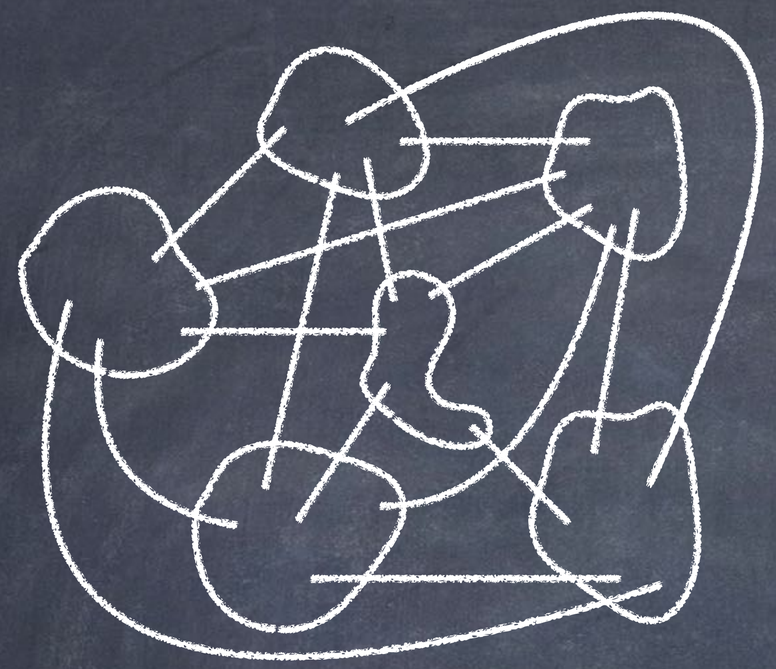
CLIQUE CONTRACTION :: Partition of V(G) s. t.

a) Each part (called witness set) is connected
b) any two witness sets are adjacent with each other.

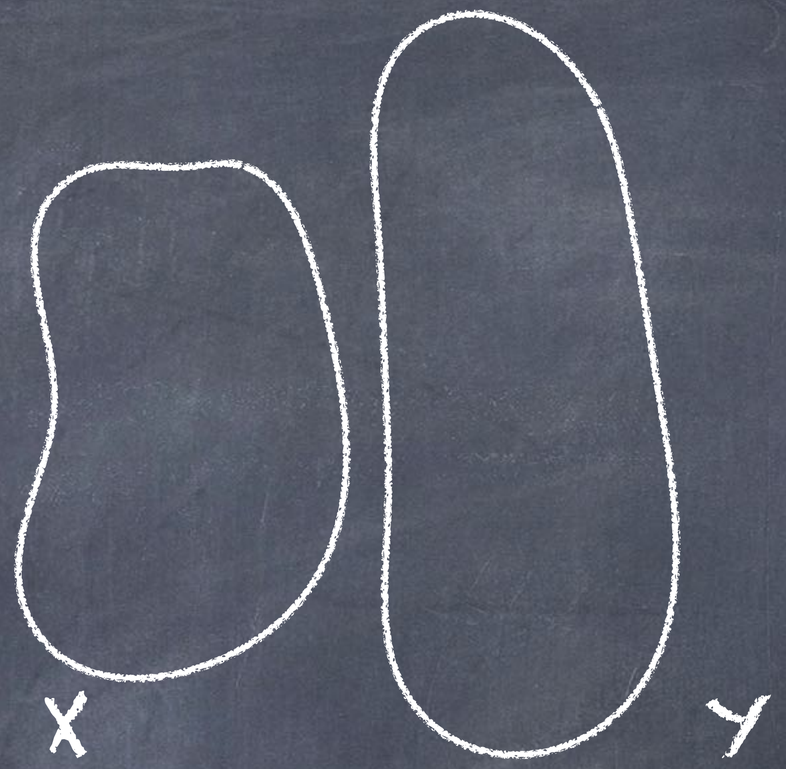


G





G

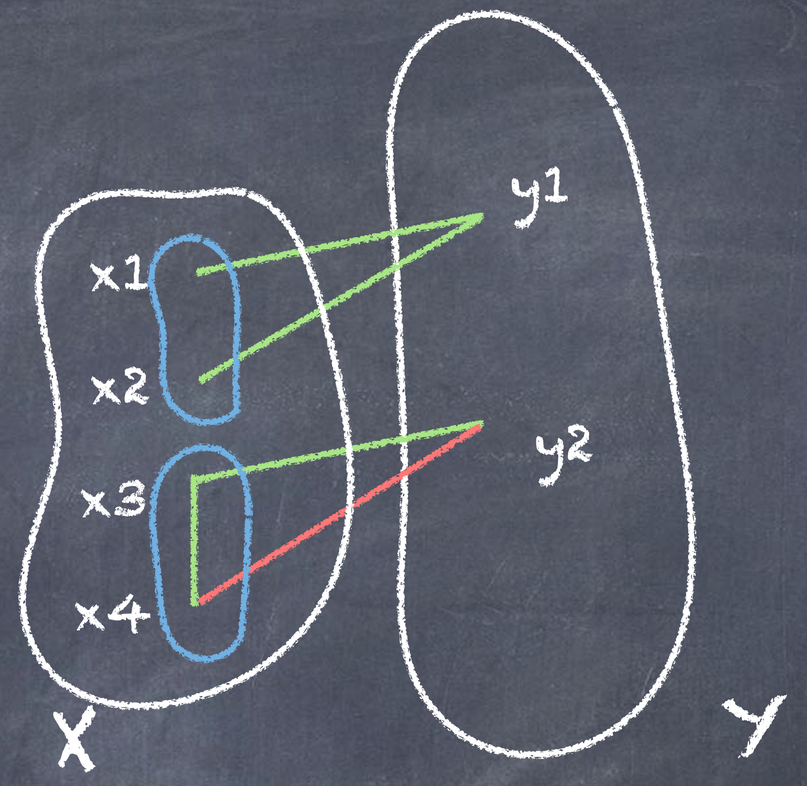


Partition (X, Y) of V(G)

- X is small
- Y induces a clique

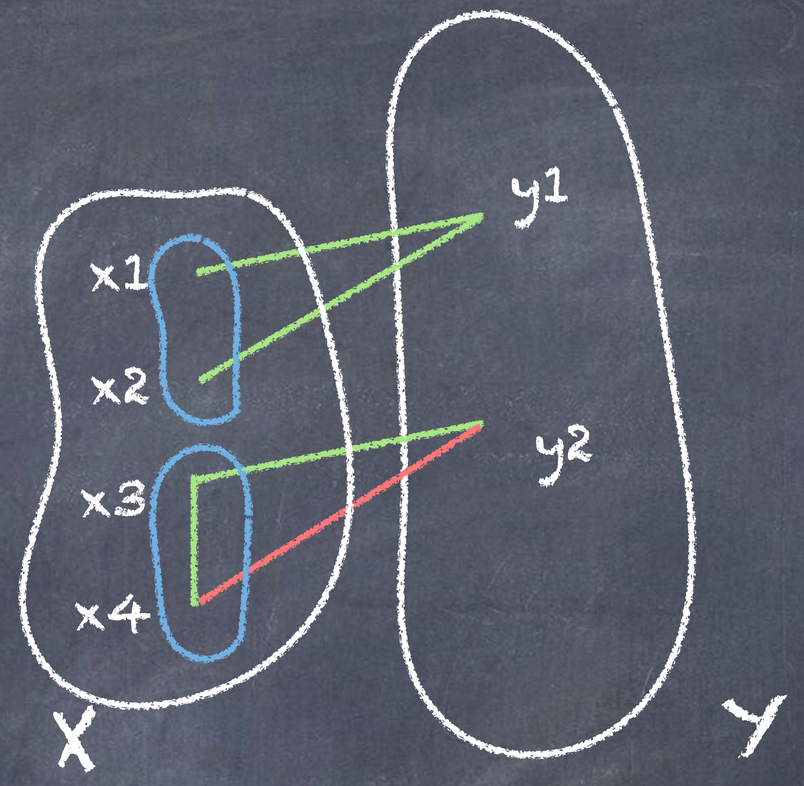
Partition (X, Y) of V(G)

- X is small
- Y induces a clique



Partition (X, Y) of V(G)

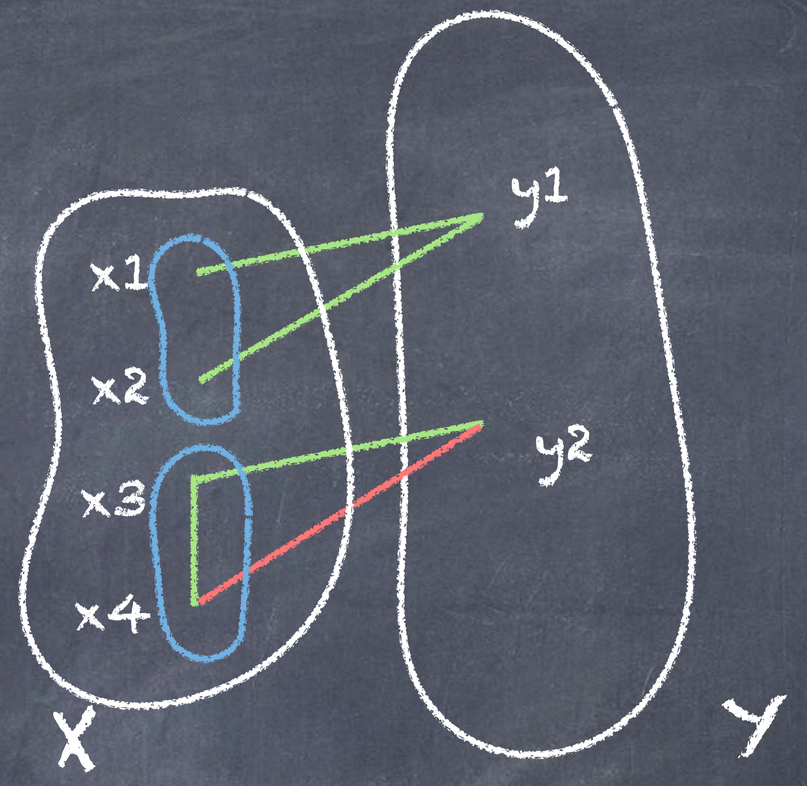
- X is small
- Y induces a clique



Roles of vertices in Y

Partition (X, Y) of V(G)

- X is small
- Y induces a clique

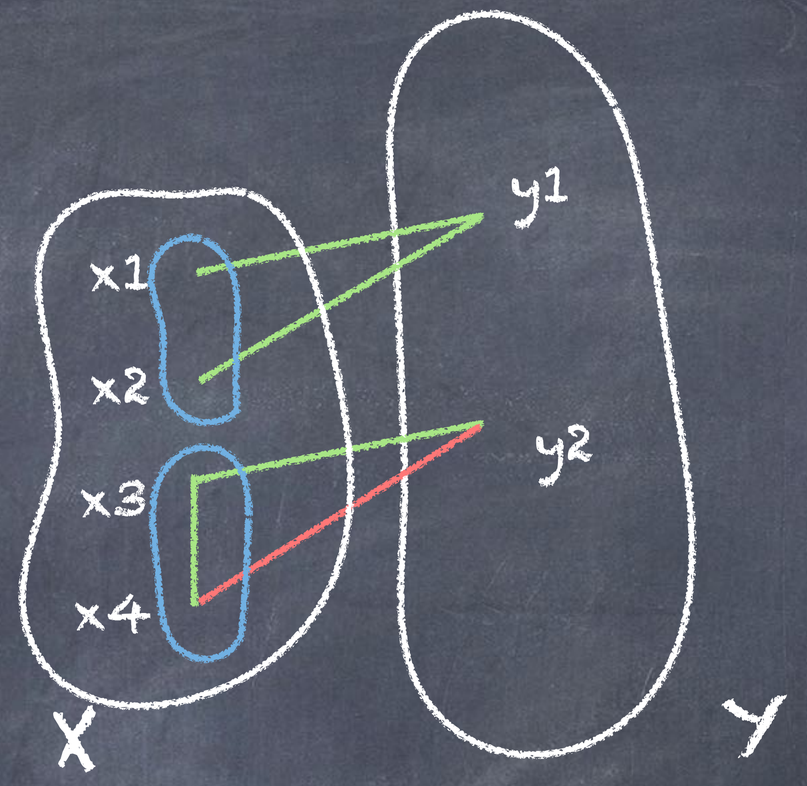


Roles of vertices in Y

– Provide connectivity to witness sets intersecting X

Partition (X, Y) of V(G)

- X is small
- Y induces a clique



Roles of vertices in Y

- Provide connectivity to witness sets intersecting X
- Dictate witness sets in X

Role of vertices in Y

- Provide connectivity to witness sets intersecting X

- Dictate witness sets in X

Role of vertices in Y

- Provide connectivity to witness sets intersecting X

  For every subset X′ of X, mark a vertex which is common neighbour.

- Dictate witness sets in X

Role of vertices in Y

- Provide connectivity to witness sets intersecting X

  For every subset X′ of X, mark a vertex which is common neighbour.

- Dictate witness sets in X

  For every subset X′ of X, mark (2k + 1) vertices which are common non-neighbour.

Role of vertices in Y

$(d + 1)/d <= \alpha$

- Provide connectivity to witness sets intersecting X

  For every subset X′ of X, mark a vertex which is common neighbour.

  For every subset X′ of X of size <= d, mark a vertex which is common neighbour.

- Dictate witness sets in X

  For every subset X′ of X, mark (2k + 1) vertices which are common non-neighbour.

  For every subset X′ of X of size <= d, mark (2k + 1) vertices which are common non-neighbour.

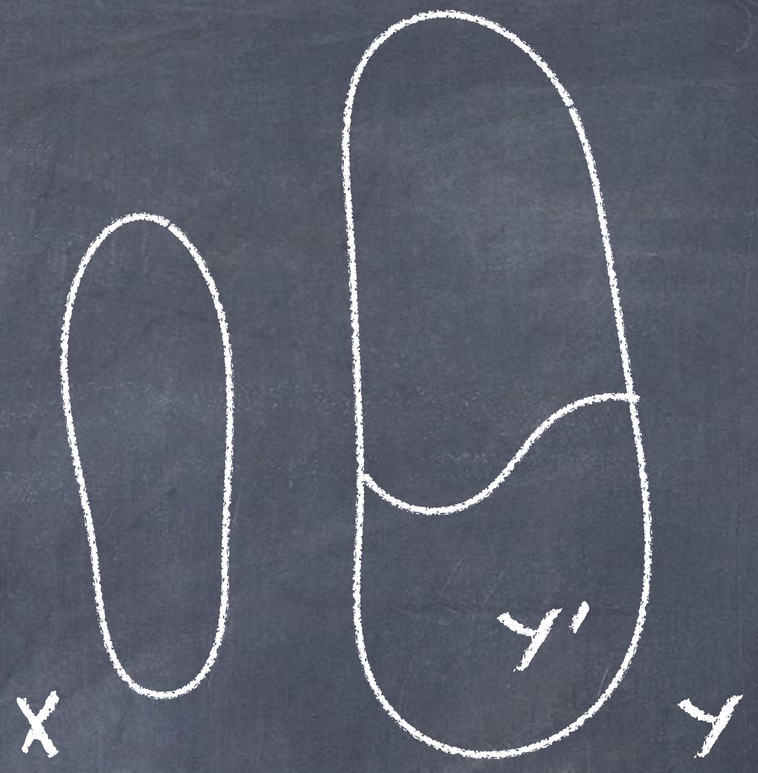
For every subset X′ of X of size <= d, mark a vertex which is common neighbour.

For every subset X′ of X of size <= d, mark (2k + 1) vertices which are common non-neighbour.

Reduction Rule 2: Mark vertices in Y and delete unmarked vertices.

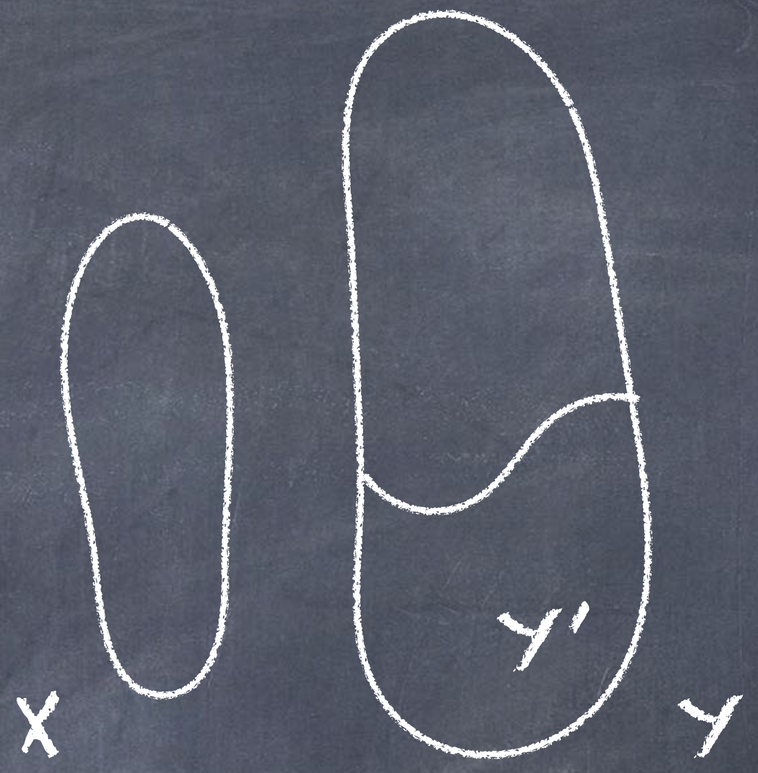
RR-2 runs in poly-time on large instances.

Y' – deleted vertices



Y' – deleted vertices

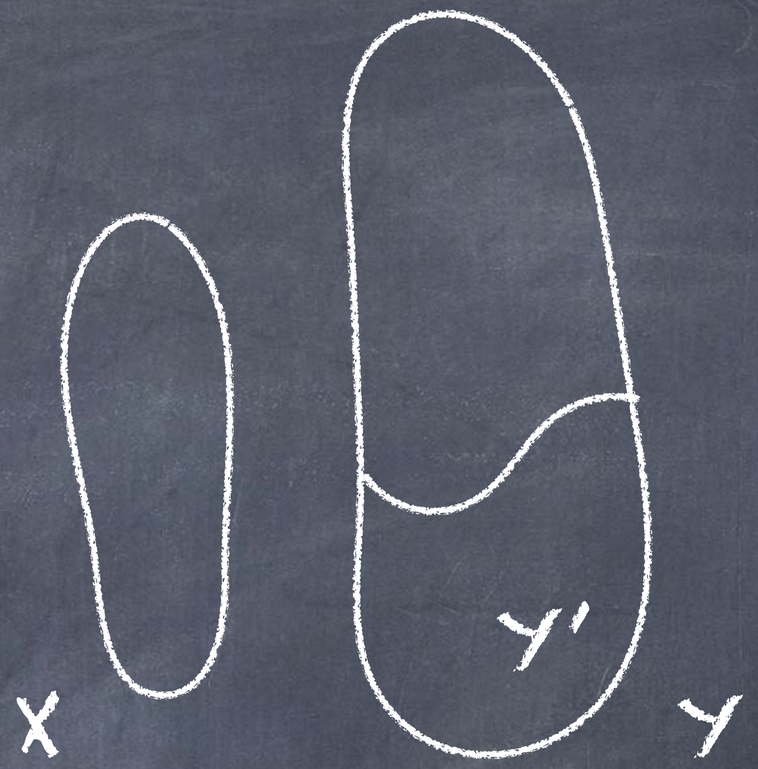
$(G \setminus Y', k)$ reduced instance.



Y' – deleted vertices

$(G \setminus Y', k)$ reduced instance.

F' – given solution for $(G \setminus Y', k)$

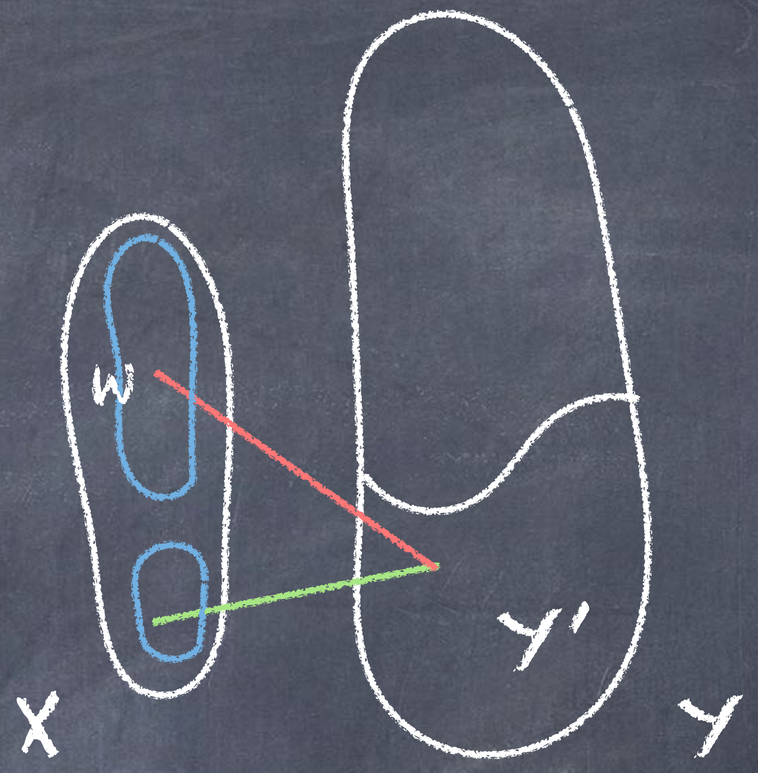


Y' – deleted vertices

$(G \setminus Y', k)$ reduced instance.

F' – given solution for $(G \setminus Y', k)$

Bad vertex in Y' = not adj.
with witness set $W \subseteq X$.



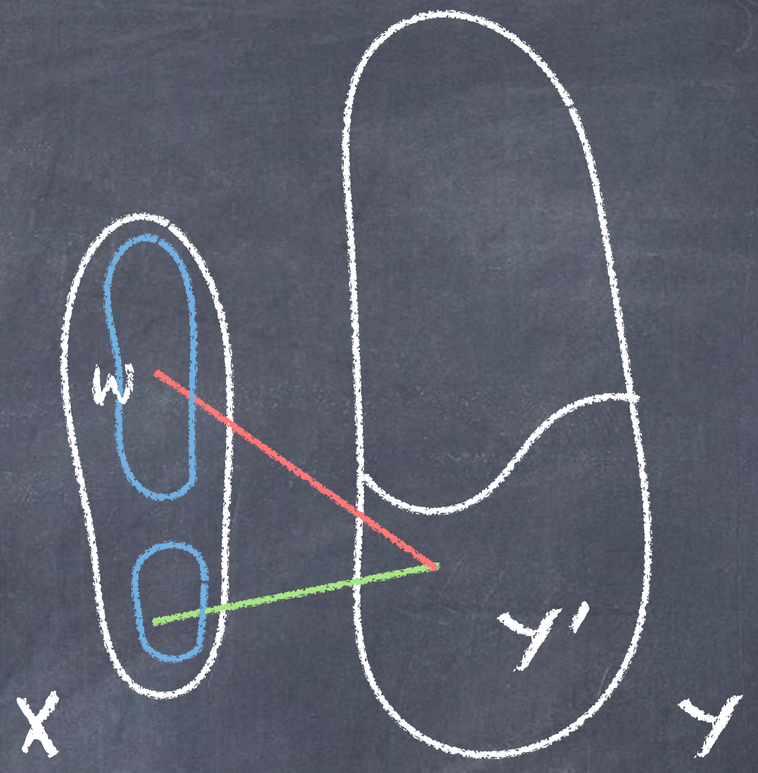
Y' – deleted vertices

$(G \setminus Y', k)$ reduced instance.

F' – given solution for $(G \setminus Y', k)$

Bad vertex in Y' = not adj.
with witness set $W \subseteq X$.

W has size at least $d + 1$



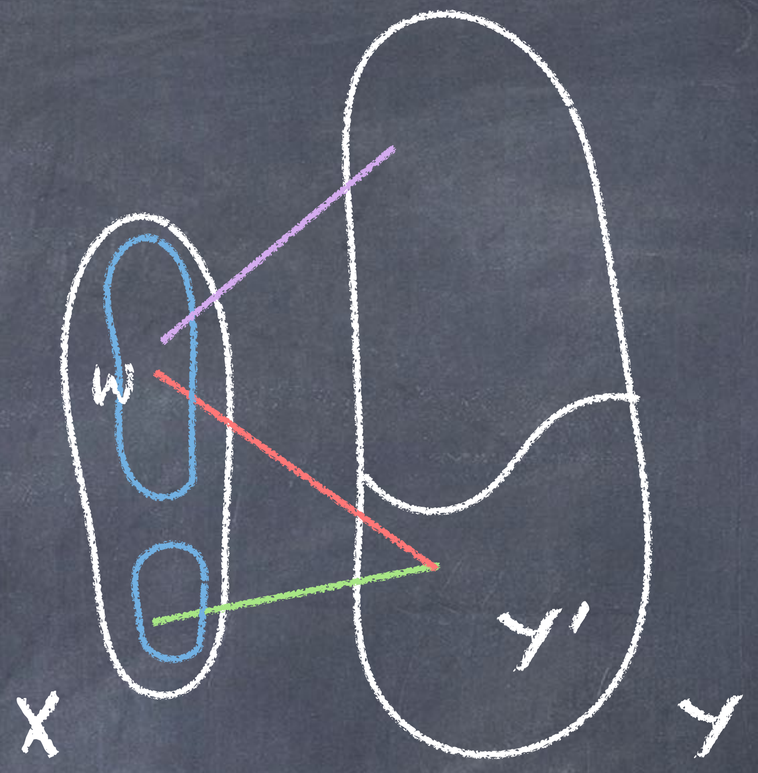
Y' – deleted vertices

$(G \setminus Y', k)$ reduced instance.

F' – given solution for $(G \setminus Y', k)$

Bad vertex in Y' = not adj.
with witness set $W \subseteq X$.

W has size at least $d + 1$



Solution Lifting Algorithm: Add a vertex in $Y \setminus Y'$ to such witness set. (Contract purple edge)

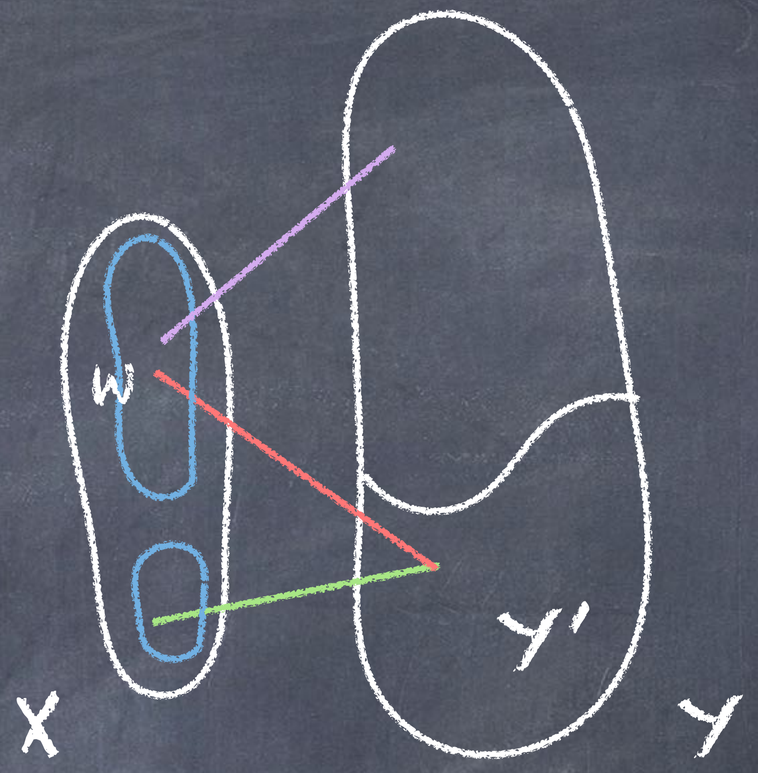
Y' – deleted vertices

$(G \setminus Y', k)$ reduced instance.

F' – given solution for $(G \setminus Y', k)$

Bad vertex in Y' = not adj.
with witness set $W \subseteq X$.

W has size at least $d + 1$



Solution Lifting Algorithm: Add a vertex in $Y \setminus Y'$ to such witness set. (Contract purple edge)

Lemma : Size of obtained solution F is $\leq \alpha |F'|$.

Lemma : Size of obtained solution F is <= α |F'|.

=

Lemma : Size of obtained solution F is <= α $|F'|$.

$$(G, k) \longrightarrow (G', k)$$

=

Lemma : Size of obtained solution F is $<= \alpha\ |F'|$.

$$(G, k) \longrightarrow (G', k)$$

Lemma-2 : $OPT(G', k) <= OPT(G, k)$.

=

Lemma : Size of obtained solution F is <= α $|F'|$.

$$(G, k) \longrightarrow (G', k)$$

Lemma-2 : $OPT(G', k) <= OPT(G, k)$.

Lemma : RR-2 is α safe.

=

Lemma : Size of obtained solution F is $<= \alpha\, |F'|$.

$$(G, k) \longrightarrow (G', k)$$

Lemma-2 : $OPT(G', k) <= OPT(G, k)$.

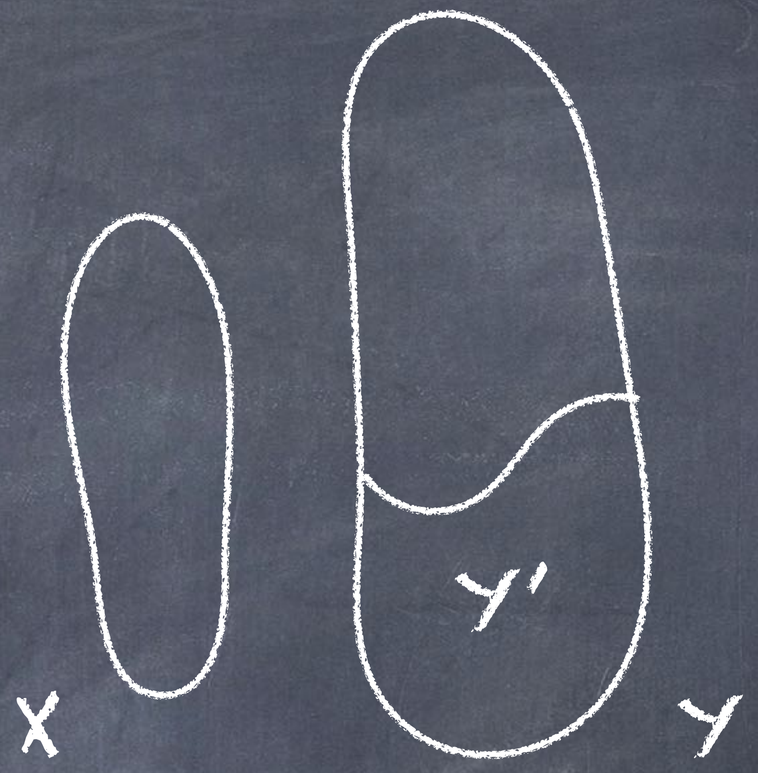
Lemma : RR-2 is α safe.

$$\frac{|F|}{OPT(G, k)} <= \alpha \frac{|F'|}{OPT(G', k)} = \alpha\, c$$

$|X| <= 4k$

$|Y \setminus Y'| <= O(k^{d+1})$

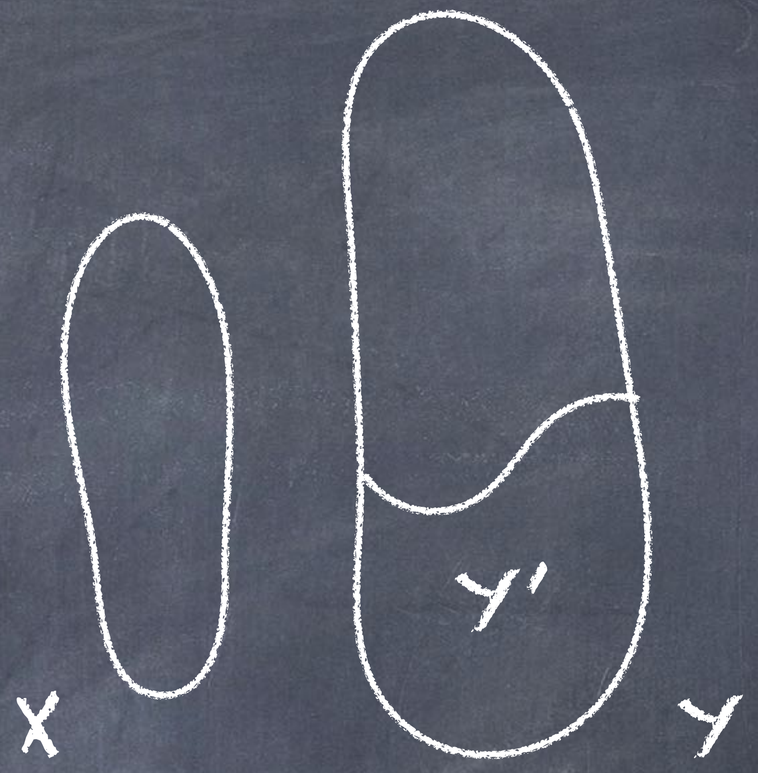
(marked vertices)



$|X| <= 4k$

$|Y \setminus Y'| <= O(k^{d+1})$

(marked vertices)



Theorem : For any $\alpha > 1$, CLIQUE CONTRACTION parameterised by the size of solution k admits an α-lossy kernel with $O(k^{d+1})$ vertices where $d = 1/\alpha$.

Parameterized Complexity and Known Results

Overview Of Our Contributions

Brief Description: Lossy Kernel for Clique Cont.

Thank you.