Automated Generation of Search Tree Algorithms for Graph Modification Problems

Jens Gramm   Jiong Guo   Falk Hüffner   Rolf Niedermeier

{gramm,guo,hueffner,niedermr}@informatik.uni-tuebingen.de.

Wilhelm-Schickard Institut für Informatik, Universität Tübingen

Search Tree Algorithms

- Search trees of bounded height (*splitting* strategy) are a standard technique for NP-hard problems.
- Evolution towards better search tree sizes:
  - MaxSat:
    $1.62^K \to 1.38^K \to 1.34^K \to 1.32^K$
  - Vertex Cover:
    $1.33^k \to 1.32^k \to 1.30^k \to 1.29^k$
- Usually based on large tedious and error-prone case distinctions
- Automating could bring:
  - rapid development
  - improved upper bounds

Splitting for VERTEX COVER

VERTEX COVER: Given a graph (V, E), find a subset $V' \subseteq V$ of vertices such that for each edge at least one of the endpoints is on V'.

Simple splitting strategy:
Choose an arbitrary edge $\{u, v\}$. Branch recursively into two cases:

- u is part of the vertex cover: Remove u and all edges adjacent to u.
- v is part of the vertex cover: Remove v and all edges adjacent to v.

Search tree size 2^n

Reduction Rules

Reduction rules replace a problem instance I with another instance I' such that

- the complexity of I' is not greater (usually smaller);
- and a solution for I can be constructed from I' in polynomial time.

Example for VERTEX COVER:

- Remove degree-0 vertices.
- For a vertex of degree one, take its neighbor into the vertex cover.
- ...

Local Case Distinction

- Consider a set of small induced subgraphs (*windows*), such that any input instance contains at least one window.
- Find a sequence of splitting and reducing operations for each case (*branching rule*).

Search Tree Algorithms

- Case Distinction
- Splitting
- Reducing

Tasks:

- Find a splitting strategy
- Find reduction rules
- Find a set of cases to distinguish
- For each case, find a good strategy of splitting and reducing

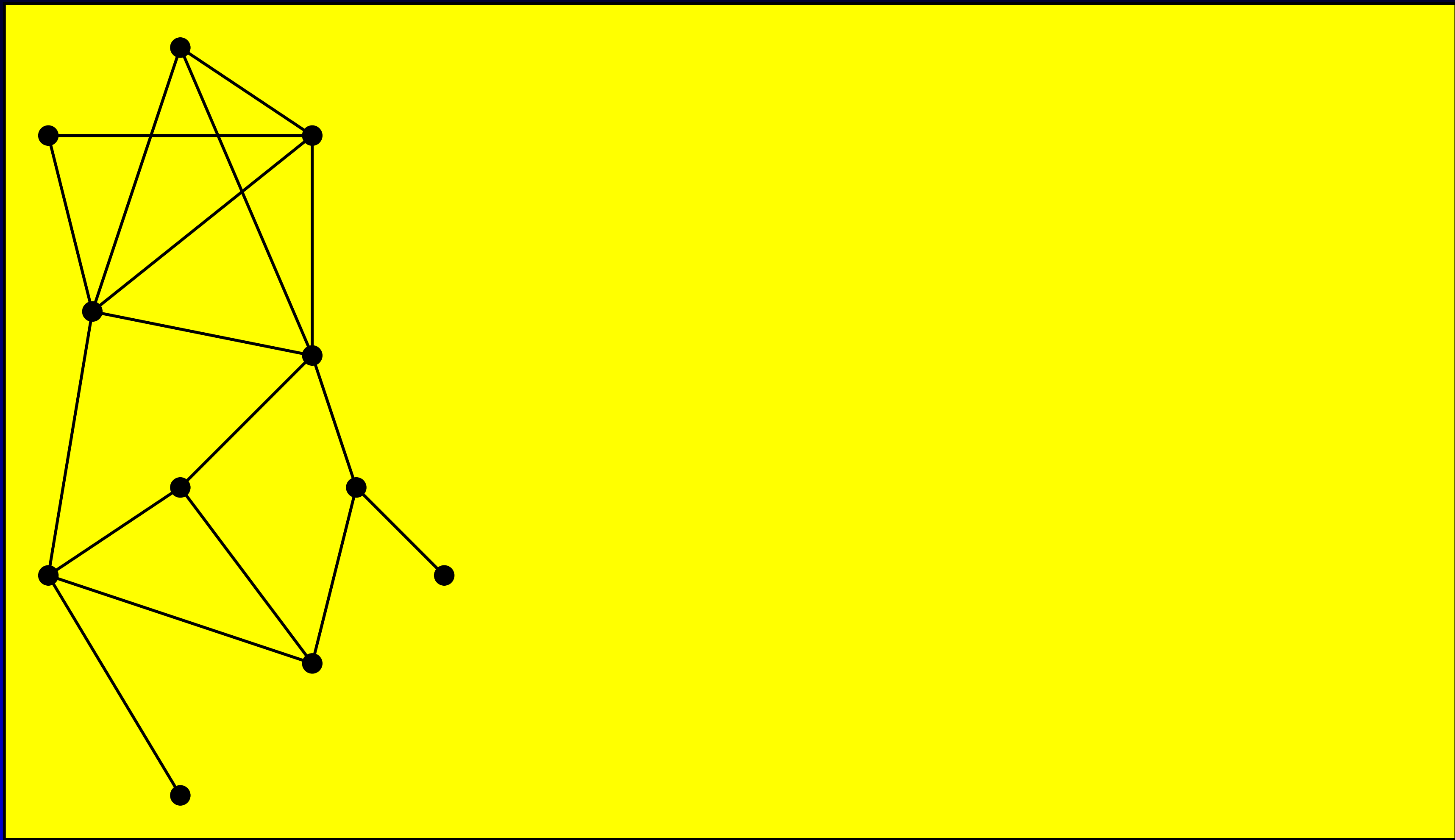
Search Tree Algorithms

- Case Distinction
- Splitting
- Reducing

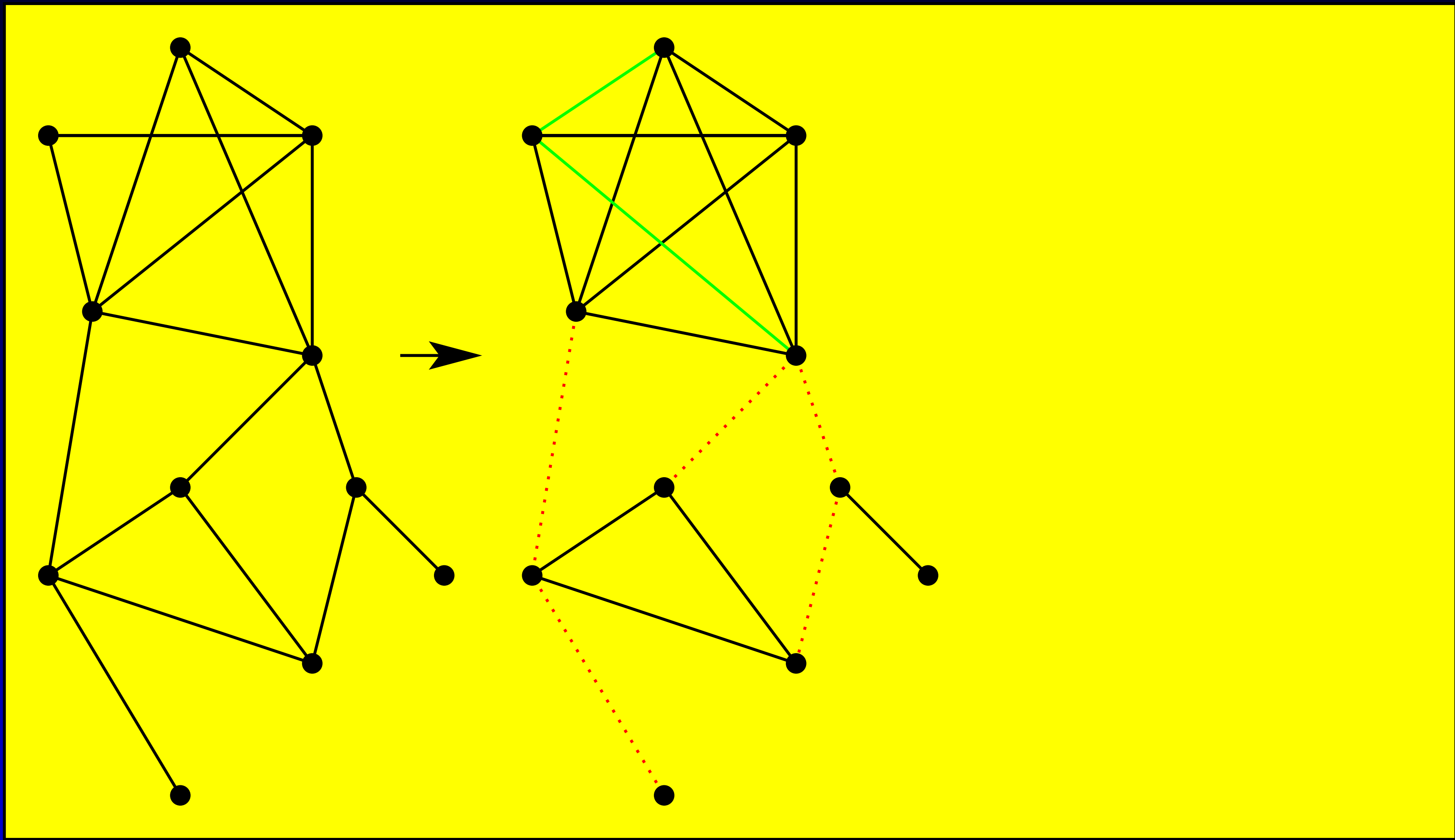
Tasks:

- Find a splitting strategy – easy
- Find reduction rules – challenging
- Find a set of cases to distinguish – tedious
- For each case, find a good strategy of splitting and reducing – tedious

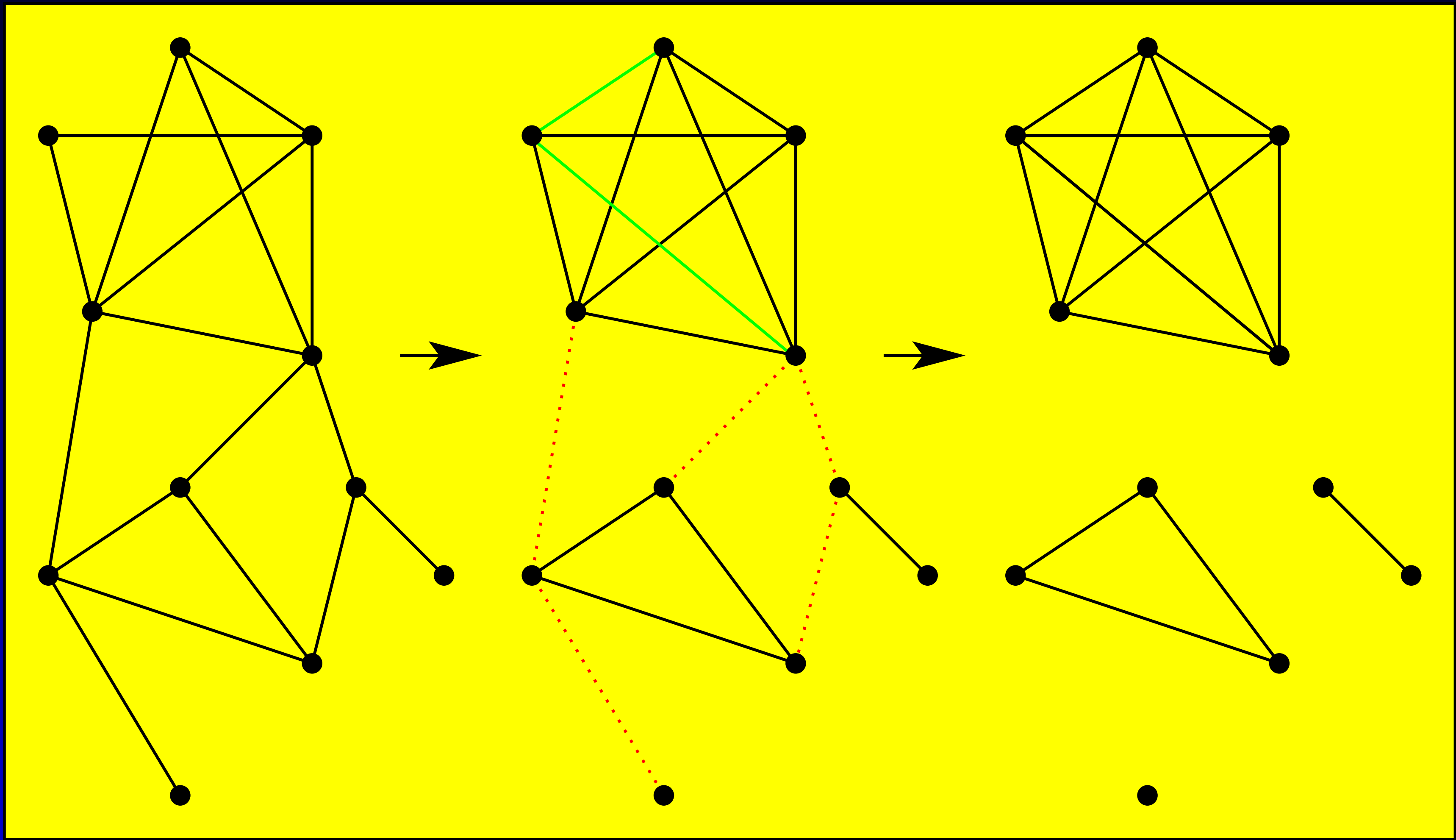
Cluster Editing



Cluster Editing



Cluster Editing



Cluster Editing

- Also known as "Correlation Clustering on complete unweighted graphs"
- Motivated from computational biology and other fields
- NP-complete
- Best known exact search tree algorithm: $O(2.27^k)$ search tree size (k: number of editing operations)

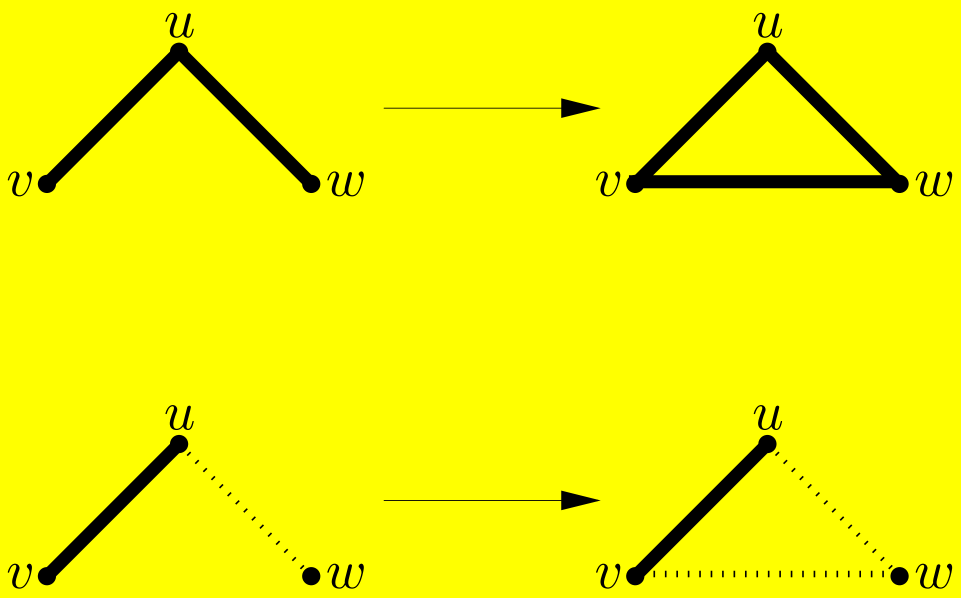
CLUSTER EDITING Splitting

Introduce two edge annotations *permanent* and *forbidden*.
Choose an arbitrary non-annotated edge $\{u, v\}$.
Branch recursively into two cases:

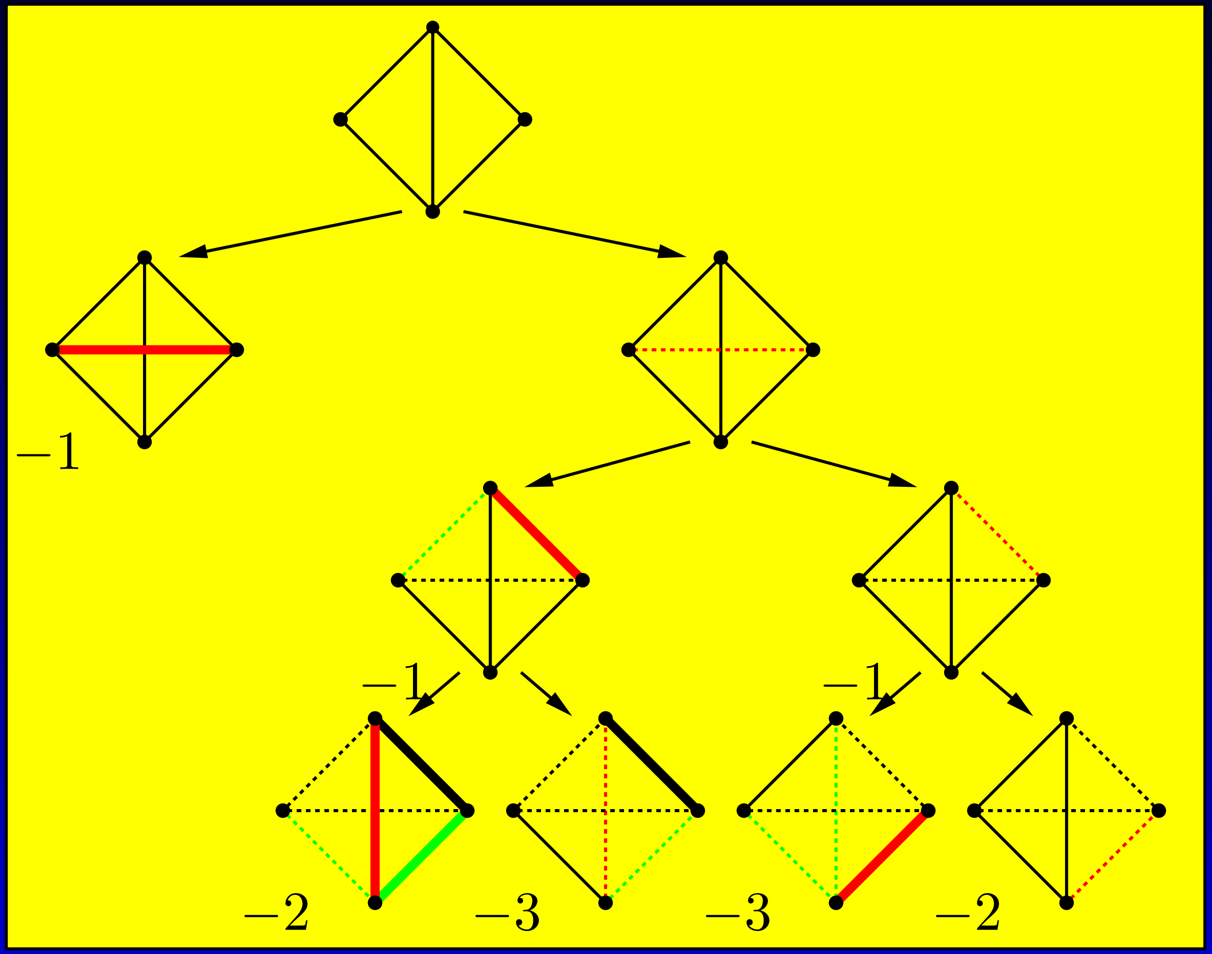
- $\{u, v\}$ is part of the clustering solution: Add $\{u, v\}$ if not present and mark it *permanent*.
- $\{u, v\}$ is not part of the clustering solution: Delete $\{u, v\}$ if present and mark it *forbidden*.

Cluster Editing Reduction

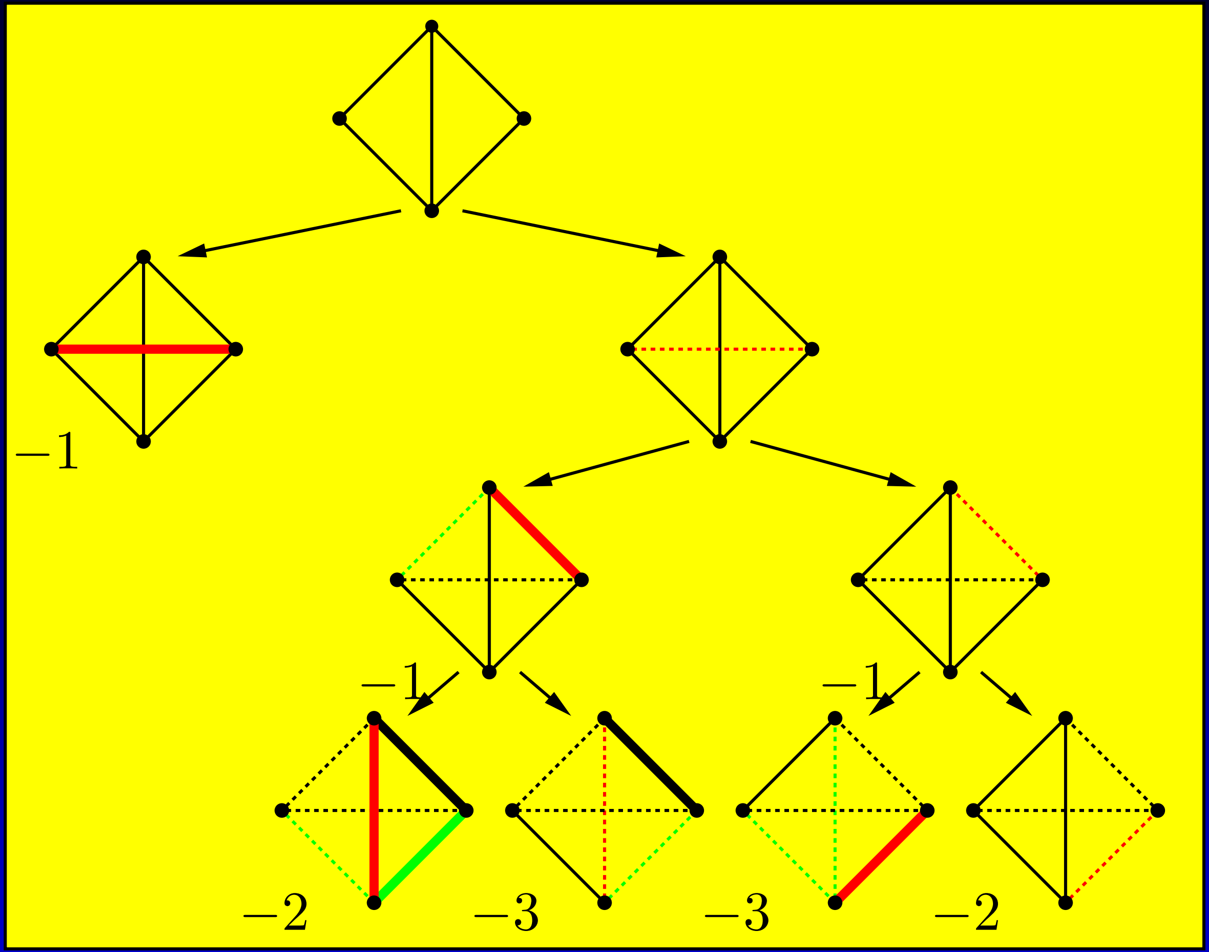
Lemma: A graph G is a cluster graph
$\iff$ G does not contain a P_3 as induced subgraph.



CLUSTER EDITING Case



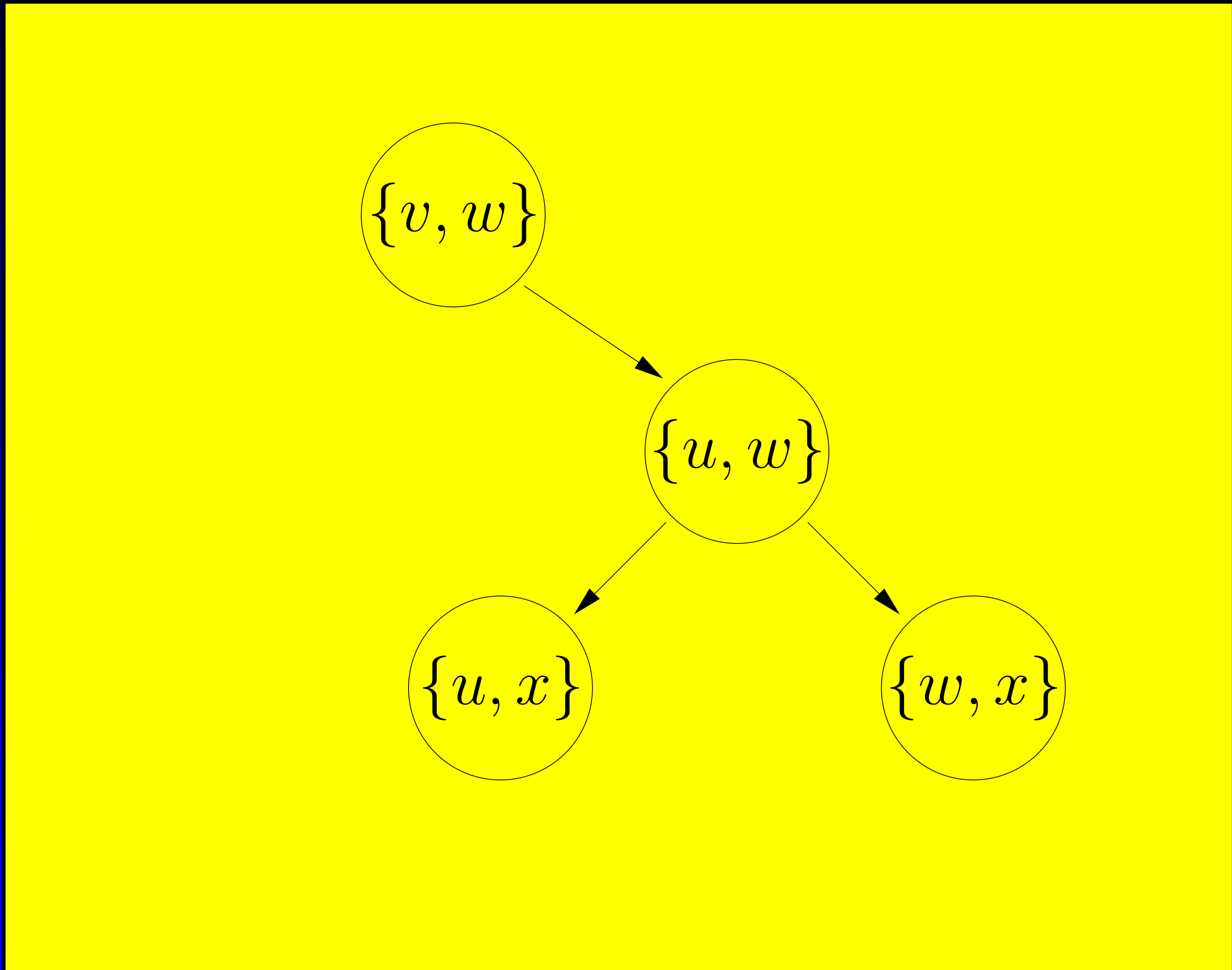
CLUSTER EDITING Case



Branching vector: $(1, 2, 3, 2, 3)$

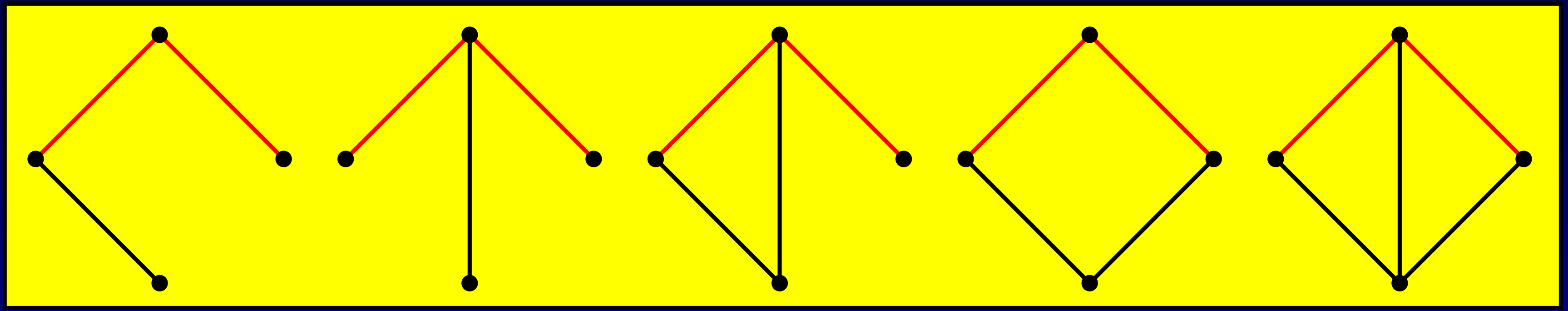
Branching number: 2.27

Branching Tree

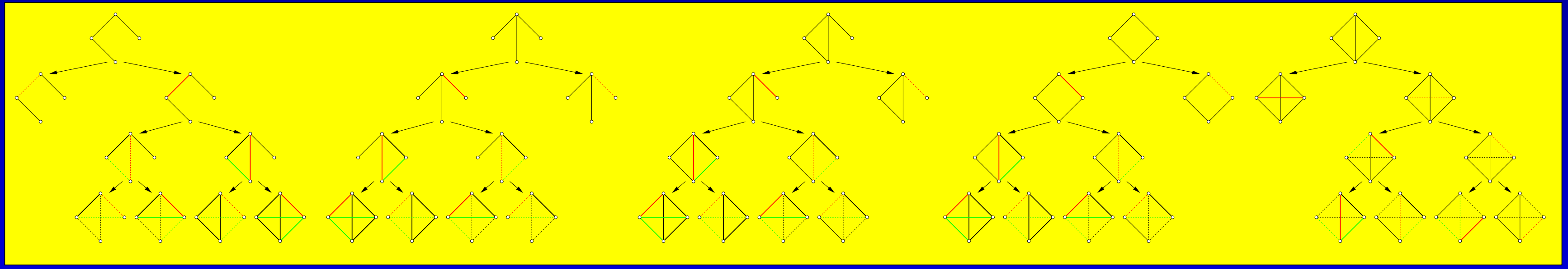


Automation

Idea: Add one vertex to a P_3, yielding a *window*, and examine all possible results:



For each possible window, try all possible branching trees, and remember the best one.



$O(2.42^k)$ $O(2.42^k)$ $O(2.27^k)$ $O(2.27^k)$ $O(2.27^k)$

Automating Results

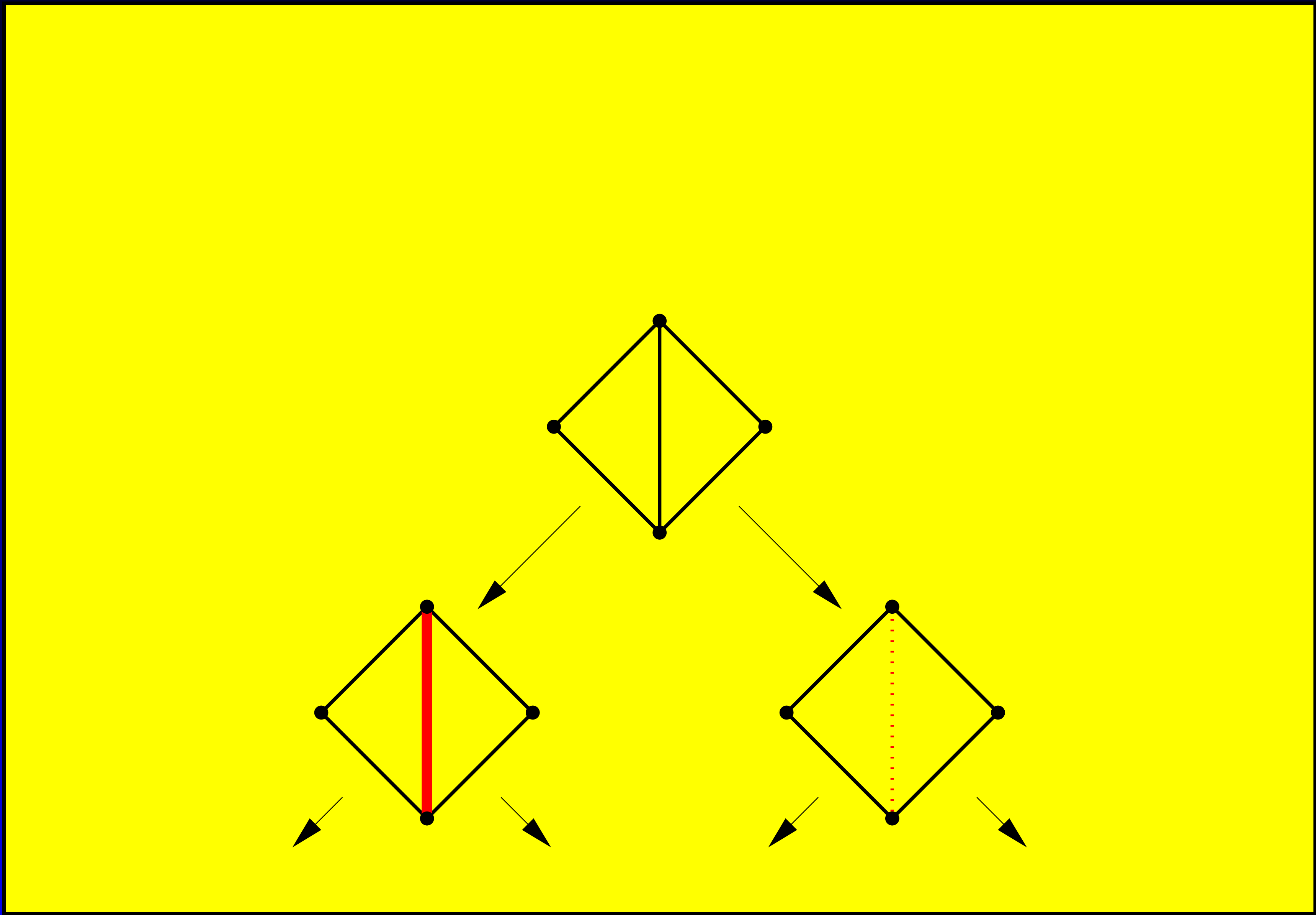
- *Automation*:
  - 5 windows analyzed
  - worst case branching number: 2.42
  - $\Rightarrow O(2.42^k)$ search tree size
- *Manual:*
  - $\Rightarrow O(2.27^k)$ search tree size

Automating

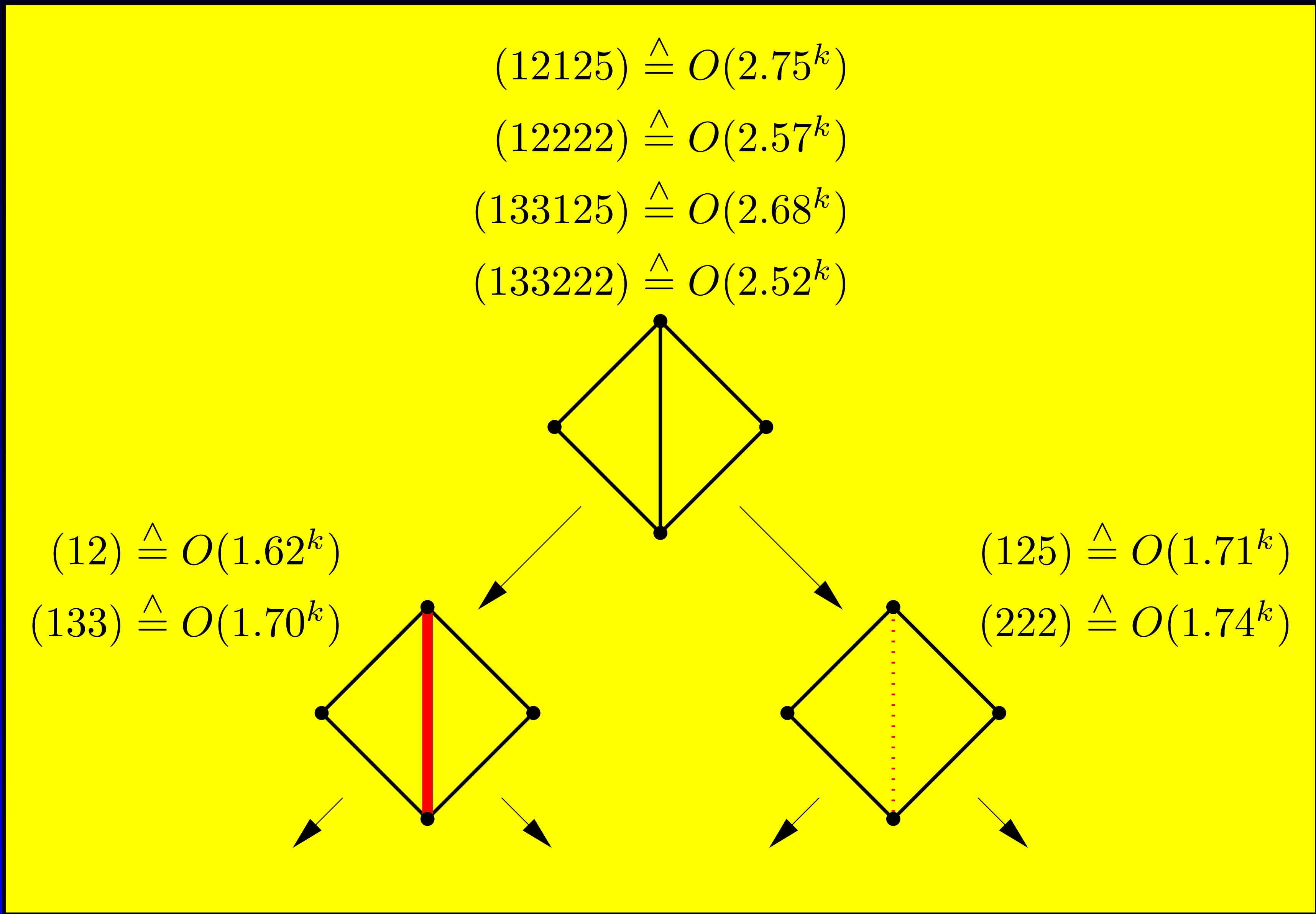
Idea: Consider larger windows.
Problem: There are very many branching trees.

Size	Windows	Branching Trees
4	5	720
5	20	3.628.800
6	111	1.307.674.368.000

Meta Search Tree



Meta Search Tree



Meta Search Tree

A *set* of branching trees has to be returned!
Optimizations:

- Some branching trees can be discarded: $(1, 2, 2, 3, 5)$ can never be better than $(1, 2, 3, 3, 6)$
- Transposition tables

Results

Method	Windows	Search Tree	Calculating Time
manual	2	$O(2.27^k)$	a few months
win4	5	$O(2.42^k)$	0.01 seconds
win5	20	$O(2.27^k)$	2 seconds
win6	111	$O(2.16^k)$	9 days

Specific Window Expansion

Up to now:

- Find a conflict triple.
- Add an arbitrary neighbor, till the windows has n vertices.
- Execute corresponding branching tree.

Idea:
Use problem specific properties to specifically add neighbors with certain properties. That way, not every window of size n is reached.

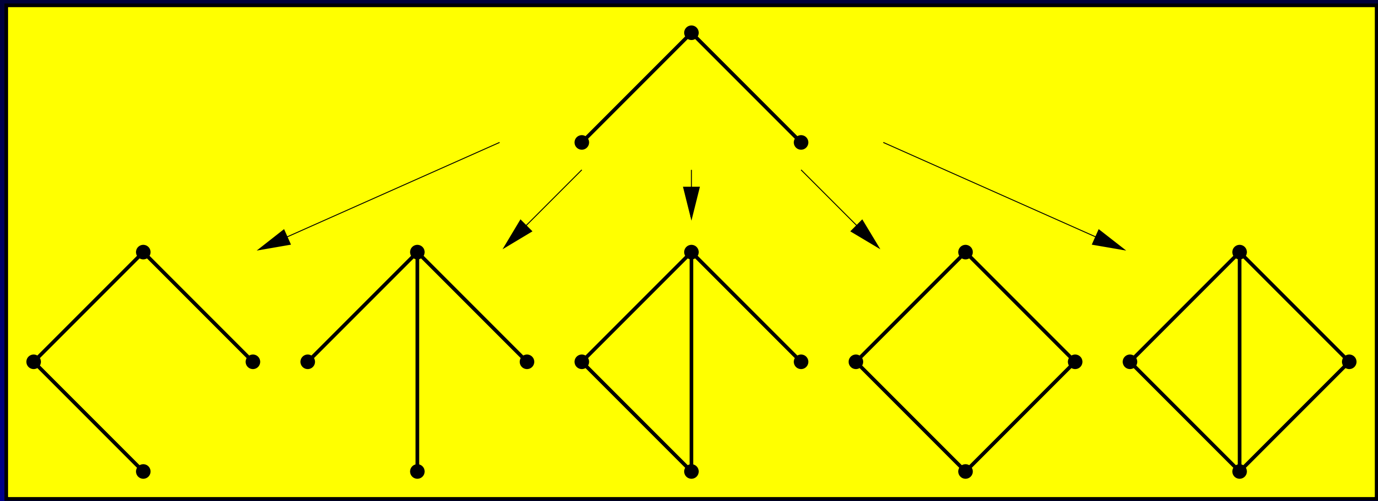
Cluster Editing Branching

Cluster Editing invariant:
Every $u, v \in E$ have a common neighbor, i.e., $\exists x$ with $\{u, x\} \in E$ and $\{v, x\} \in E$.

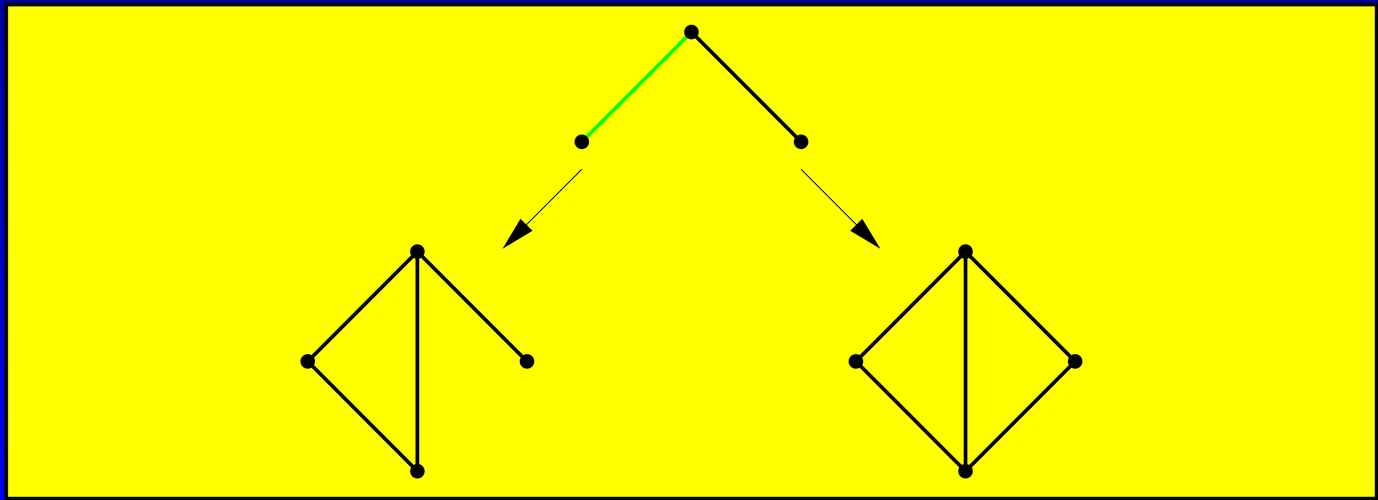
(Otherwise, we can apply a special, more efficient branching.)

Window Expansion

Trivial:



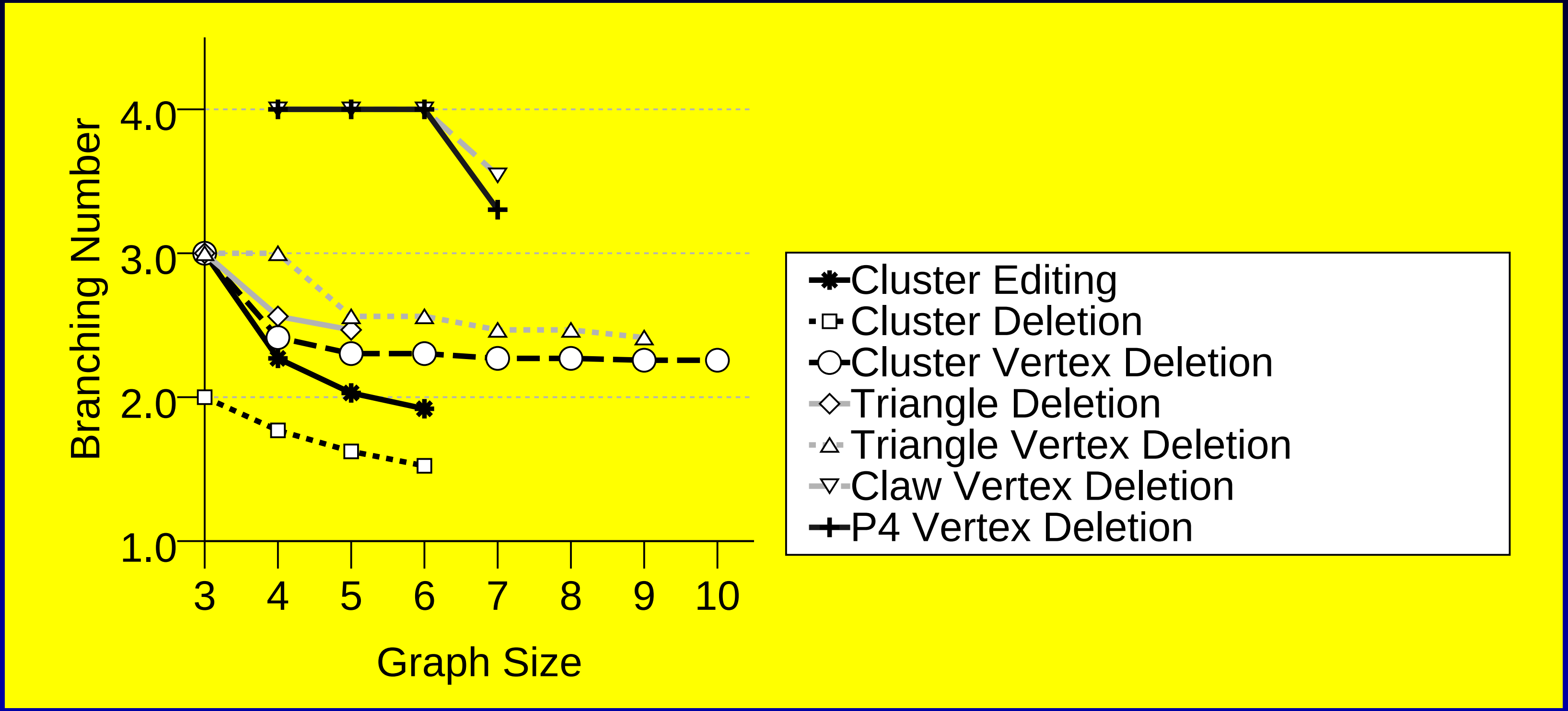
With CLUSTER EDITING special branching:



Results

Method	Cases	Search Tree	Calculating Time
manual	2	$O(2.27^k)$	48 hours
win4	5	$O(2.42^k)$	0.01 seconds
win5	20	$O(2.27^k)$	2 seconds
win6	111	$O(2.16^k)$	9 days
expand4	6	$O(2.27^k)$	0.01 seconds
expand5	26	$O(2.03^k)$	3 seconds
expand6	137	$O(1.92^k)$	9 days

Other Problems



General Problem Description

To apply this scheme to a problem, we need:

- Annotations (optional)
- Splitting
- Expansion rules
- Reduction rules

Examples:

- VERTEX COVER
- MAXSAT, X3SAT
- BOUNDED DEGREE DOMINATING SET

But probably not:

- TRAVELING SALESMAN

Open questions

- Generate “executable search tree algorithm code” and test it
- Heuristics
- Improvements for graph modification problems
- Application to further problems