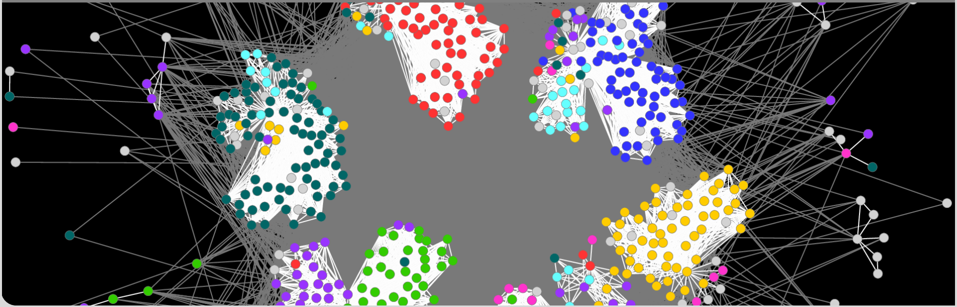
Graph Editing Problems and their Application in Graph Clustering

CALDAM Indo-German Pre-Conference School on Algorithms and Combinatorics

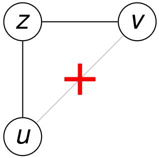
Dorothea Wagner | February 13-14, 2017

KARLSRUHE INSTITUTE OF TECHNOLOGY – INSTITUTE OF THEORETICAL INFORMATICS – GROUP ALGORITHMICS

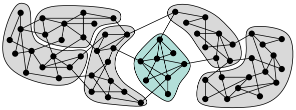


Overview

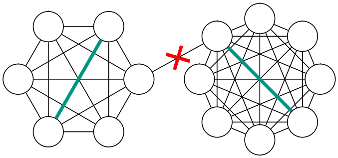
- Introduction to graph editing



- Introduction to graph clustering



- Cluster editing

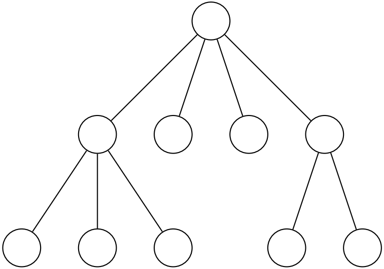


Graph Classes

Graph $\mathcal{G}$ is part of a graph class if it fulfills certain properties.

Examples:

- Trees
- Planar graphs
- Chordal graphs

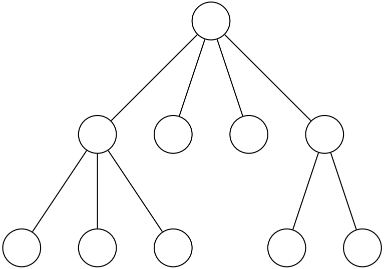


Graph Classes

Graph $\mathcal{G}$ is part of a graph class if it fulfills certain properties.

Examples:

- Trees
- Planar graphs
- Chordal graphs



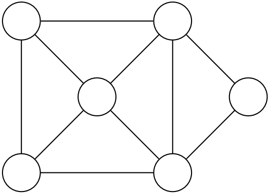
Graph Classes

Karlsruhe Institute of Technology

Graph $\mathcal{G}$ is part of a graph class if it fulfills certain properties.

Examples:

- Trees
- Planar graphs
- Chordal graphs



Institute of Theoretical Informatics
Group Algorithmics

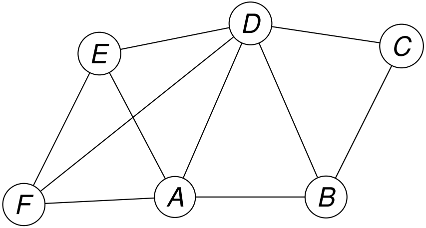
Graph Classes

Karlsruhe Institute of Technology

Graph $\mathcal{G}$ is part of a graph class if it fulfills certain properties.

Examples:

- Trees
- Planar graphs
- Chordal graphs

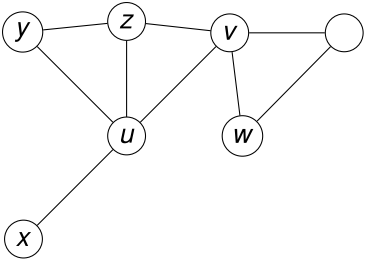


Editing

Given a graph $\mathcal{G}$ – what is the smallest number of operations that need to be applied such that $\mathcal{G}$ is part of a graph class $\mathcal{H}$?

Possible operations:

- Delete an edge
- Insert an edge
- Delete a node
- Insert a node

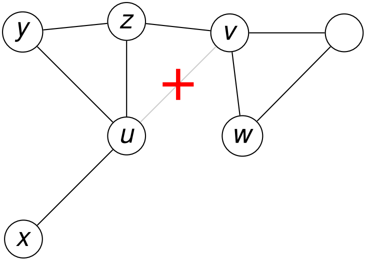


Editing

Given a graph $\mathcal{G}$ – what is the smallest number of operations that need to be applied such that $\mathcal{G}$ is part of a graph class $\mathcal{H}$?

Possible operations:

- Delete an edge
- Insert an edge
- Delete a node
- Insert a node

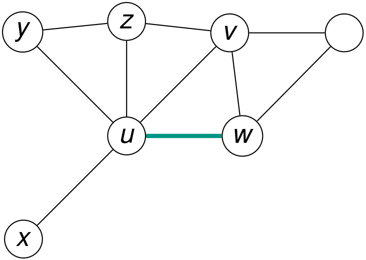


Editing

Given a graph $\mathcal{G}$ – what is the smallest number of operations that need to be applied such that $\mathcal{G}$ is part of a graph class $\mathcal{H}$?

Possible operations:

- Delete an edge
- Insert an edge
- Delete a node
- Insert a node

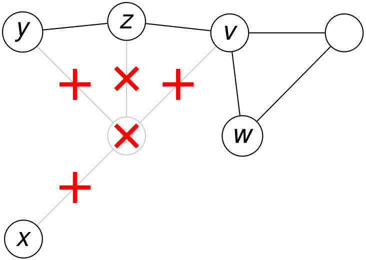


Editing

Given a graph $\mathcal{G}$ – what is the smallest number of operations that need to be applied such that $\mathcal{G}$ is part of a graph class $\mathcal{H}$?

Possible operations:

- Delete an edge
- Insert an edge
- Delete a node
- Insert a node

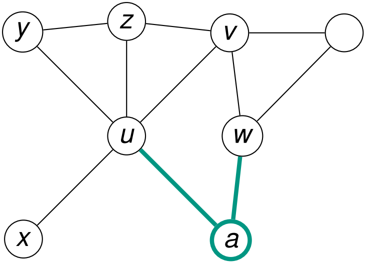


Editing

Given a graph $\mathcal{G}$ – what is the smallest number of operations that need to be applied such that $\mathcal{G}$ is part of a graph class $\mathcal{H}$?

Possible operations:

- Delete an edge
- Insert an edge
- Delete a node
- Insert a node

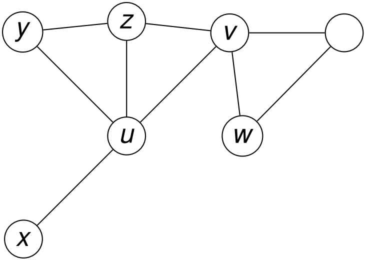


Editing

Given a graph $\mathcal{G}$ – what is the smallest number of operations that need to be applied such that $\mathcal{G}$ is part of a graph class $\mathcal{H}$?

Possible operations:

- Delete an edge
- Insert an edge
- Delete a node
- Insert a node



Can assign costs to operations – minimize sum of costs.

Example: Spanning Forest

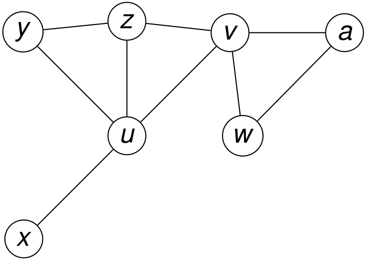
Spanning Forest

Given a graph $G = (V, E)$, find a maximal set $F \subseteq E$ such that $H = (V, F)$ is a forest.

Equivalent:
find a set of minimum size of edges X such that $G \setminus X$ is a forest.

As editing problem:

- Operations: edge deletion
- Target class: forest



$O(m + n)$ (e.g. BFS, DFS)

Example: Spanning Forest

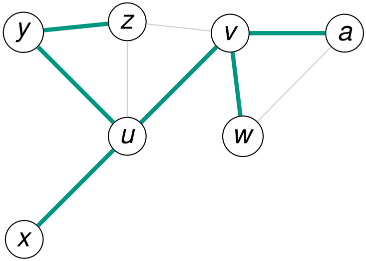
Spanning Forest

Given a graph $G = (V, E)$, find a maximal set $F \subseteq E$ such that $H = (V, F)$ is a forest.

Equivalent:
find a set of minimum size of edges X such that $G \setminus X$ is a forest.

As editing problem:

- Operations: edge deletion
- Target class: forest



$O(m + n)$ (e.g. BFS, DFS)

Example: Spanning Forest

Spanning Forest

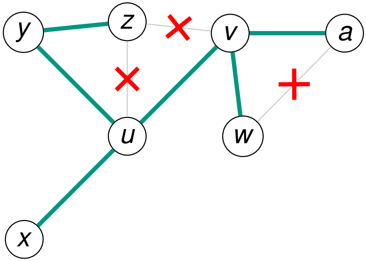
Given a graph $G = (V, E)$, find a maximal set $F \subseteq E$ such that $H = (V, F)$ is a forest.

Equivalent:
find a set of minimum size of edges X such that $G \setminus X$ is a forest.

As editing problem:

- Operations: edge deletion
- Target class: forest

$O(m + n)$ (e.g. BFS, DFS)



Example: Maximum Spanning Forest

Maximum Spanning Forest

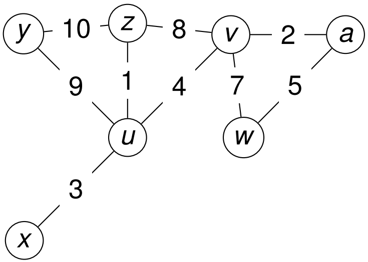
Given a weighted graph $G = (V, E, \omega)$, find a set $F \subseteq E$ of maximum weight such that $H = (V, F)$ is a forest.

Equivalent:
find a set of minimum weight of edges X such that $G \setminus X$ is a forest.

As editing problem:

- Operations: edge deletion
- Target class: forest
- Costs: edge weights

Kruskal's algorithm: $O(m \log n)$



Example: Maximum Spanning Forest

Maximum Spanning Forest

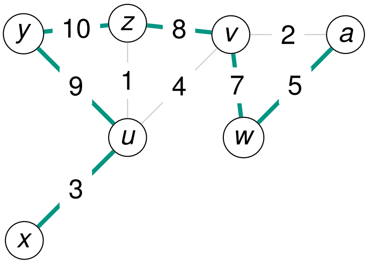
Given a weighted graph $G = (V, E, \omega)$, find a set $F \subseteq E$ of maximum weight such that $H = (V, F)$ is a forest.

Equivalent:
find a set of minimum weight of edges X such that $G \setminus X$ is a forest.

As editing problem:

- Operations: edge deletion
- Target class: forest
- Costs: edge weights

Kruskal's algorithm: $O(m \log n)$



Example: Maximum Spanning Forest

Maximum Spanning Forest

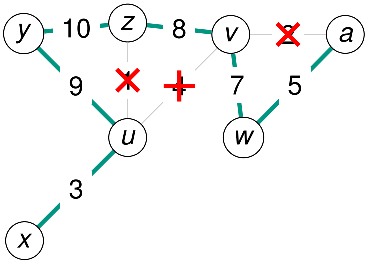
Given a weighted graph $G = (V, E, \omega)$, find a set $F \subseteq E$ of maximum weight such that $H = (V, F)$ is a forest.

Equivalent:
find a set of minimum weight of edges X such that $G \setminus X$ is a forest.

As editing problem:

- Operations: edge deletion
- Target class: forest
- Costs: edge weights

Kruskal's algorithm: $O(m \log n)$



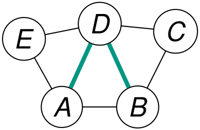
Example: Chordal Completion

Chordal Graph

A graph $G = (V, E)$ is chordal iff all cycles of four or more vertices have a chord, i.e., an edge that is not part of the cycle but connects two vertices of it.

Minimum Chordal Completion:

- Operation: edge insertion
- Target class: chordal graphs



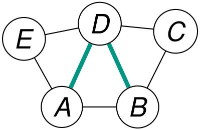
Example: Chordal Completion

Chordal Graph

A graph $G = (V, E)$ is chordal iff all cycles of four or more vertices have a chord, i.e., an edge that is not part of the cycle but connects two vertices of it.

Minimum Chordal Completion:

- Operation: edge insertion
- Target class: chordal graphs



Treewidth

One less than the size of the largest clique in a chordal completion with smallest clique number.

Example: Independent Set

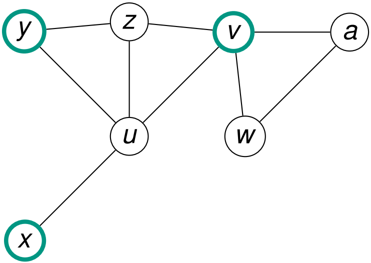
Maximum Independent Set

Given a graph $G = (V, E)$, find a set $I \subseteq V$ of maximum size such that the graph induced by I has no edges.

Equivalent:
Find a minimum set of nodes X such that $G \setminus X$ has no edges.

As editing problem:

- Allowed operations: node deletions
- Target class: graphs without edges



NP-complete.

Example: Independent Set

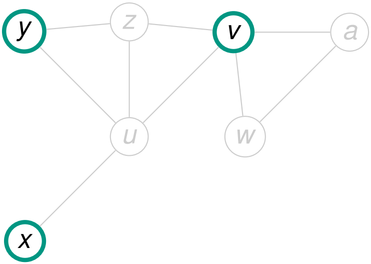
Maximum Independent Set

Given a graph $G = (V, E)$, find a set $I \subseteq V$ of maximum size such that the graph induced by I has no edges.

Equivalent:
Find a minimum set of nodes X such that $G \setminus X$ has no edges.

As editing problem:

- Allowed operations: node deletions
- Target class: graphs without edges



NP-complete.

Example: Independent Set

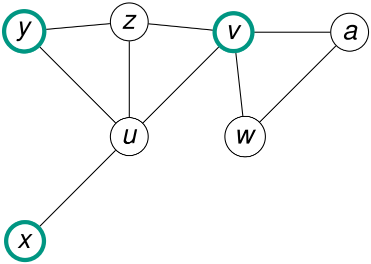
Maximum Independent Set

Given a graph $G = (V, E)$, find a set $I \subseteq V$ of maximum size such that the graph induced by I has no edges.

Equivalent:
Find a minimum set of nodes X such that $G \setminus X$ has no edges.

As editing problem:

- Allowed operations: node deletions
- Target class: graphs without edges



NP-complete.

Example: Independent Set

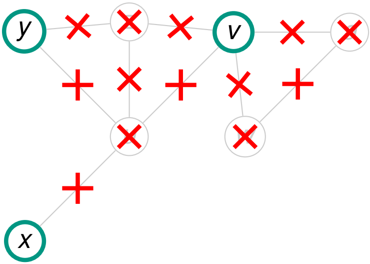
Maximum Independent Set

Given a graph $G = (V, E)$, find a set $I \subseteq V$ of maximum size such that the graph induced by I has no edges.

Equivalent:
Find a minimum set of nodes X such that $G \setminus X$ has no edges.

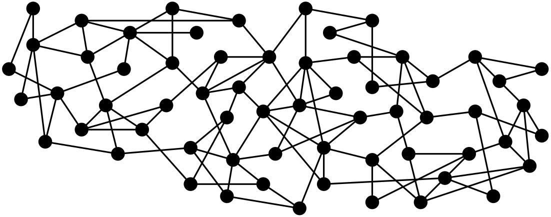
As editing problem:

- Allowed operations: node deletions
- Target class: graphs without edges



NP-complete.

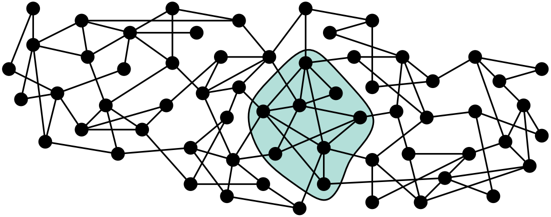
Graph Clustering: A First Intuition



- graph with a particular edge structure

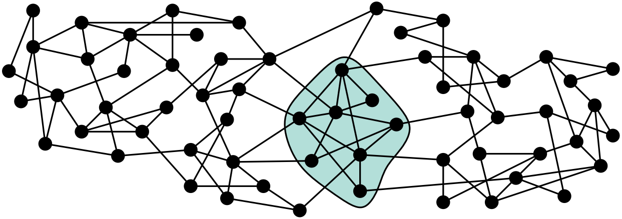
Institute of Theoretical Informatics
Group Algorithmics

Graph Clustering: A First Intuition



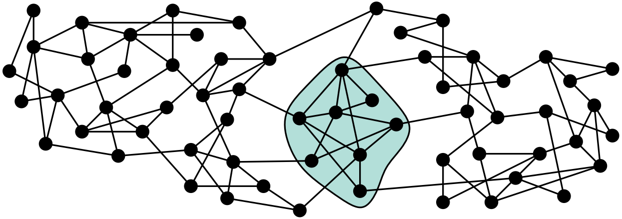
- graph with a particular edge structure
- identify subgraphs that are significantly dense

Graph Clustering: A First Intuition



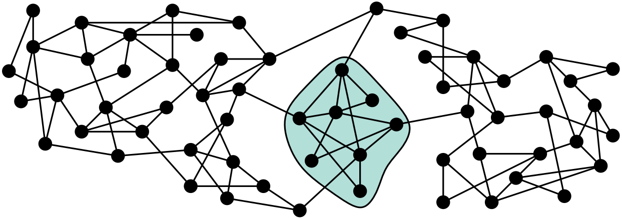
- graph with a particular edge structure
- identify subgraphs that are significantly dense
- external sparsity $\rightarrow$ more significant

Graph Clustering: A First Intuition



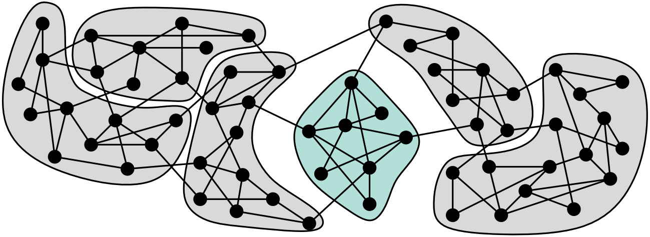
- graph with a particular edge structure
- identify subgraphs that are significantly dense
- external sparsity $\rightarrow$ more significant

Graph Clustering: A First Intuition



- graph with a particular edge structure
- identify subgraphs that are significantly dense
- external sparsity $\rightarrow$ more significant

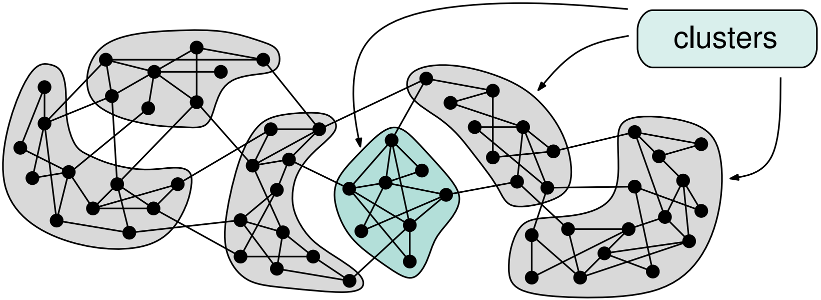
Graph Clustering: A First Intuition



- graph with a particular edge structure
- identify subgraphs that are significantly dense
- external sparsity $\rightarrow$ more significant

$\longrightarrow$ decomposition into dense subgraphs

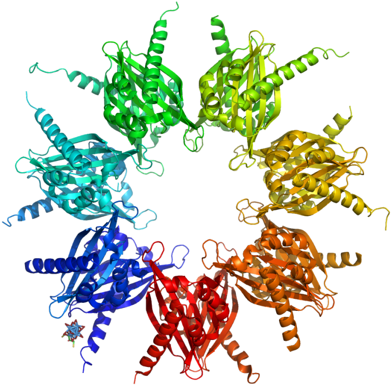
Graph Clustering: A First Intuition



- graph with a particular edge structure
- identify subgraphs that are significantly dense
- external sparsity $\rightarrow$ more significant

$\longrightarrow$ decomposition into dense subgraphs (= Clustering)

Applications in Biology

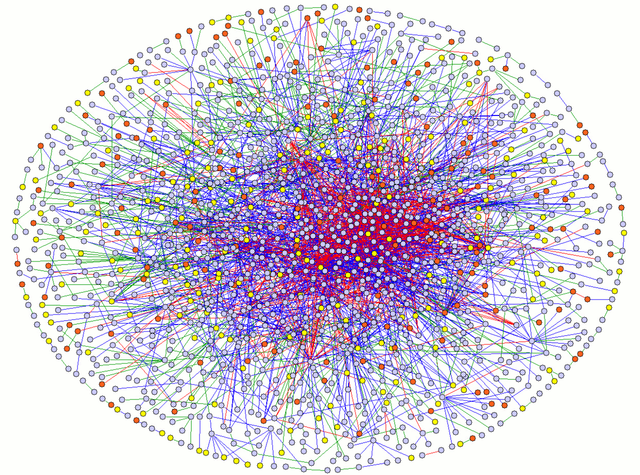


molecular structure of a protein

(Ca^{2+}/Calmodulin-dependent kinase II (CaMKII)
source: protein database www.rcsb.org)

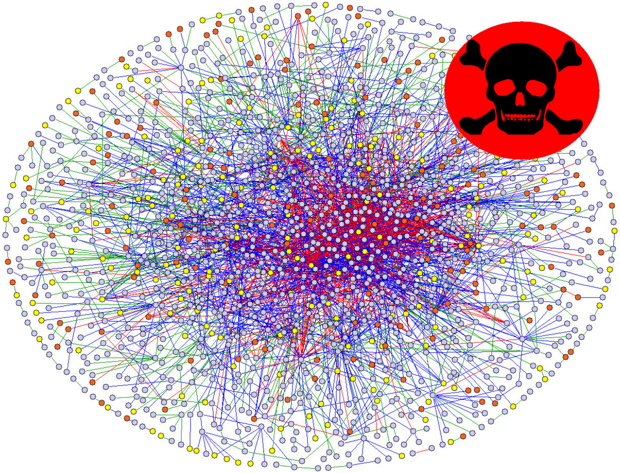
cluster $\approx$ functional unit (*domain*) of a protein

Applications in Biology



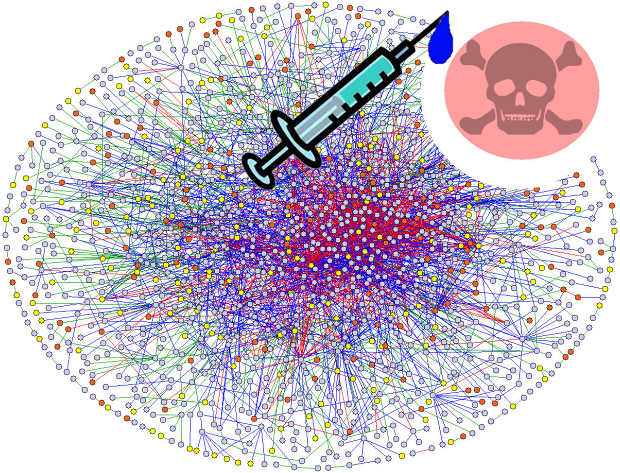
protein interactions
(source: Max-Delbrück-Centre for molecular medicine, www.mdc-berlin.de)

Applications in Biology



protein interactions
(source: Max-Delbrück-Centre for molecular medicine, www.mdc-berlin.de)

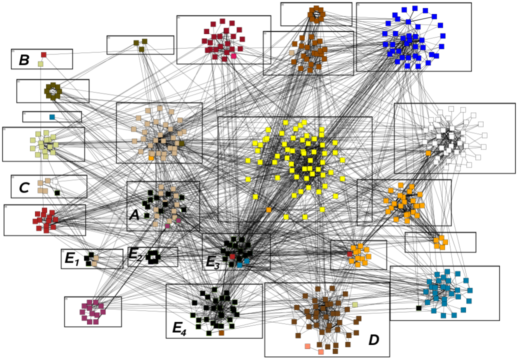
Applications in Biology



protein interactions
(source: Max-Delbrück-Centre for molecular medicine, www.mdc-berlin.de)

cluster ≈ isolatable seat of disease

Applications in Social Network Analysis



static snapshot: edges = 3 months of emails

From Intuition to Formalization

Paradigm of Graph Clustering

Intra-cluster density vs. **inter-cluster sparsity**

$\Downarrow$

Mathematical Formalization

- quality measures for clusterings
- models for communities – cliques, quasi-cliques, . . .

Many exist, optimization generally (NP-)hard

There is no single, universally best strategy

Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

- more intra-edges $\Rightarrow$ higher quality

Institute of Theoretical Informatics
Group Algorithmics

Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

- more intra-edges $\Rightarrow$ higher quality
- less inter-edges $\Rightarrow$ higher quality

Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

- more intra-edges $\Rightarrow$ higher quality
- less inter-edges $\Rightarrow$ higher quality
- cliques must never be separated

Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

- more intra-edges $\Rightarrow$ higher quality
- less inter-edges $\Rightarrow$ higher quality
- cliques must never be separated
- clusters must be connected

Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

- more intra-edges $\Rightarrow$ higher quality
- less inter-edges $\Rightarrow$ higher quality
- cliques must never be separated
- clusters must be connected
- random clusterings should have bad quality

Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

- more intra-edges $\Rightarrow$ higher quality
- less inter-edges $\Rightarrow$ higher quality
- cliques must never be separated
- clusters must be connected
- random clusterings should have bad quality
- disjoint cliques should approach maximum quality

Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

- more intra-edges $\Rightarrow$ higher quality
- less inter-edges $\Rightarrow$ higher quality
- cliques must never be separated
- clusters must be connected
- random clusterings should have bad quality
- disjoint cliques should approach maximum quality
- locality of the measure (being better/worse in one part does not depend on what is done in other part of graph)

Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

- more intra-edges $\Rightarrow$ higher quality
- less inter-edges $\Rightarrow$ higher quality
- cliques must never be separated
- clusters must be connected
- random clusterings should have bad quality
- disjoint cliques should approach maximum quality
- locality of the measure (being better/worse in one part does not depend on what is done in other part of graph)
- double the instance, what should happen . . . same result

Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

- more intra-edges $\Rightarrow$ higher quality
- less inter-edges $\Rightarrow$ higher quality
- cliques must never be separated
- clusters must be connected
- random clusterings should have bad quality
- disjoint cliques should approach maximum quality
- locality of the measure (being better/worse in one part does not depend on what is done in other part of graph)
- double the instance, what should happen . . . same result
- comparable results across instances

Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

- more intra-edges $\Rightarrow$ higher quality
- less inter-edges $\Rightarrow$ higher quality
- cliques must never be separated
- clusters must be connected
- random clusterings should have bad quality
- disjoint cliques should approach maximum quality
- locality of the measure (being better/worse in one part does not depend on what is done in other part of graph)
- double the instance, what should happen . . . same result
- comparable results across instances
- fulfill the desiderata of the application

Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

- more intra-edges $\Rightarrow$ higher quality
- less inter-edges $\Rightarrow$ higher quality
- cliques must never be separated
- clusters must be connected
- random clusterings should have bad quality
- disjoint cliques should approach maximum quality
- locality of the measure (being better/worse in one part does not depend on what is done in other part of graph)
- double the instance, what should happen ... same result
- comparable results across instances
- fulfill the desiderata of the application
- ...

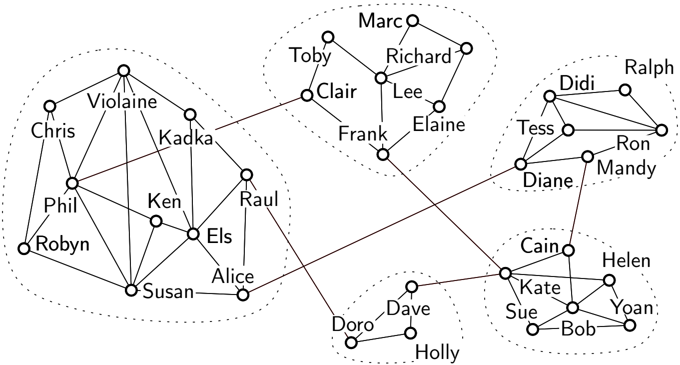
Postulations to a Measure

Given a graph G and a clustering $\mathcal{C}$, a *quality measure* should behave as follows:

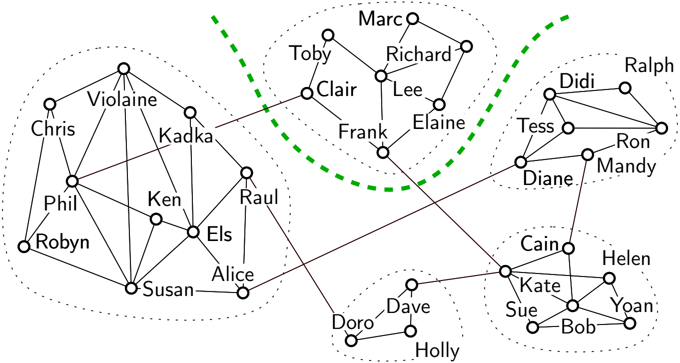
- more intra-edges $\Rightarrow$ higher quality
- less inter-edges $\Rightarrow$ higher quality
- cliques must never be separated
- clusters must be connected
- random clusterings should have bad quality
- disjoint cliques should approach maximum quality
- locality of the measure (being better/worse in one part does not depend on what is done in other part of graph)
- double the instance, what should happen . . . same result
- comparable results across instances
- fulfill the desiderata of the application
- . . .

Kleinberg: An impossibility theorem for clustering [Kle02]

Formalization via Bottleneck



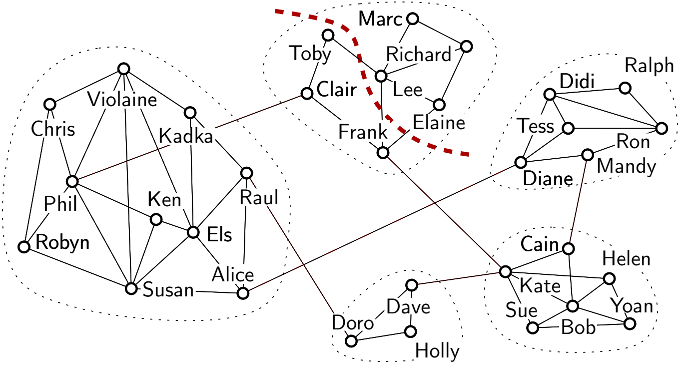
Formalization via Bottleneck



Quality of the clustering, upper cluster:

- inter-cluster sparsity: 2 *edges for cutting off* 7 *nodes* **(cheap)**

Formalization via Bottleneck



Quality of the clustering, upper cluster:

- inter-cluster sparsity: 2 *edges for cutting off* 7 *nodes* **(cheap)**
- intra-cluster density: best addit. cut:
  3 *edges for cutting off* 4 *nodes* **(expensive)**

Examples: Conductance, Expansion

conductance of a cut $(C, V \setminus C)$:

$$\varphi(C, V \setminus C) := \frac{\omega(E(C, V \setminus C))}{\min\left\{\sum_{v \in C} \omega(v), \sum_{v \in V \setminus C} \omega(v)\right\}}$$

(i.e.: thickness of bottleneck which cuts off C)

inter-cluster conductance $(\mathcal{C}) := 1 - \max_{C \in \mathcal{C}} \ \varphi(C, V \setminus C)$

(i.e.: $1-$ worst bottleneck induced by some $C \in \mathcal{C}$)

intra-cluster conductance $(\mathcal{C}) := \quad \min_{C \in \mathcal{C}} \ \min_{P \uplus Q = C} \ \varphi_{|C}(P, Q)$

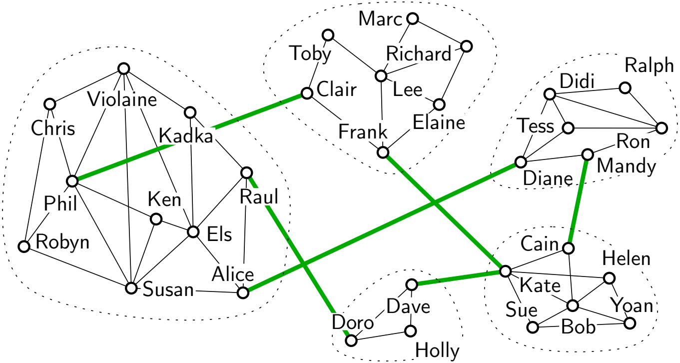
(i.e.: best bottleneck still left uncut inside some $C \in \mathcal{C}$)

expansion of a cut $(C, V \setminus C)$:

$$\psi(C, V \setminus C) := \frac{\omega(E(C, V \setminus C))}{\min\left\{|C|, |V \setminus C|\right\}}$$

(i.e.: in φ, replace $\omega(v)$ by 1; *intra-* and *inter-cluster expansion* analogously)

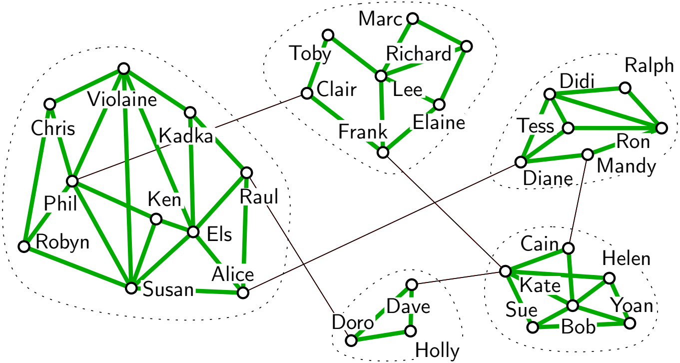
Formalization: Counting Edges



Measuring clustering quality by counting edges:

- inter-cluster sparsity: 6 *edges of ca. 800 node pairs* **(few)**

Formalization: Counting Edges

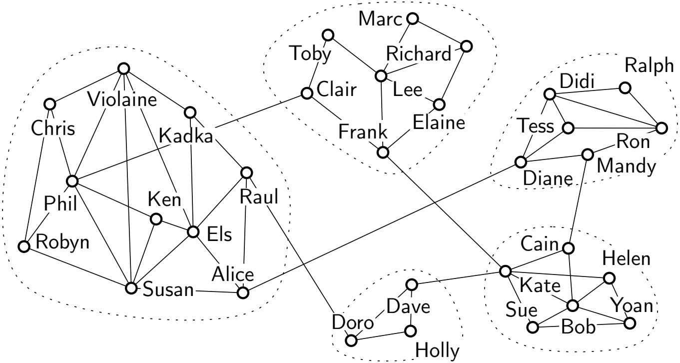


Measuring clustering quality by counting edges:

- inter-cluster sparsity: 6 *edges of ca. 800 node pairs* **(few)**
- intra-cluster density: 53 *edges of* 99 *node pairs* **(many)**

Institute of Theoretical Informatics
Group Algorithmics

Example: Coverage

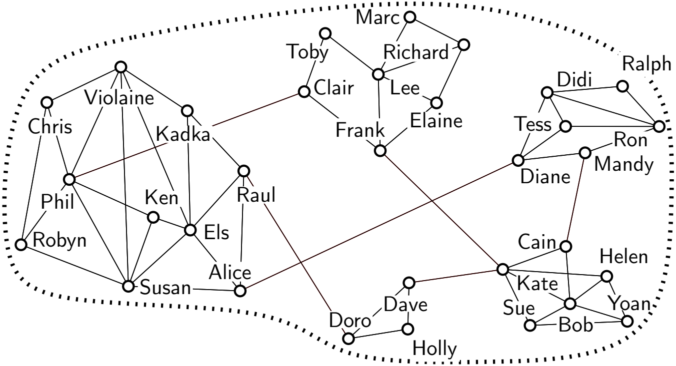


- *coverage:* $\mathrm{cov}(\mathcal{C}) := \dfrac{\text{\# intra-cluster edges}}{\text{\# edges}} \approx 0.9$

  (i.e.: fraction of covered edges)

Institute of Theoretical Informatics
Group Algorithmics

Example: Coverage



- *coverage:* $\text{cov}(\mathcal{C}) := \dfrac{\text{\# intra-cluster edges}}{\text{\# edges}} \approx 0.9$

  (i.e.: fraction of covered edges)

- **only one cluster $\Rightarrow$ *coverage* = 1.0**

A Promising Remedy

Karlsruhe Institute of Technology

"... if we subtract from [coverage] the ***expected*** *value [...], we do get a useful measure."* [NG04]

A Promising Remedy

"... if we subtract from [coverage] the ***expected*** *value [...], we do get a useful measure."* [NG04]

Modularity

$$\text{mod}(\mathcal{C}) := \quad \mathit{cov}(\mathcal{C}) \quad - \mathbb{E}(\mathit{cov}(\mathcal{C}))$$

$$= \frac{\text{\# intra-cluster edges}}{|\#\mathit{edges}|} - \frac{1}{4|\#\mathit{edges}|^2} \sum_{C \in \mathcal{C}} \left(\sum_{v \in C} \deg(v) \right)^2$$

A Promising Remedy

"... if we subtract from [coverage] the ***expected*** *value [...], we do get a useful measure."* [NG04]

Modularity

$$\mathrm{mod}(\mathcal{C}) := \mathit{cov}(\mathcal{C}) - \mathbb{E}(\mathit{cov}(\mathcal{C}))$$

$$= \frac{\#\text{ intra-cluster edges}}{|\#\mathit{edges}|} - \frac{1}{4|\#\mathit{edges}|^2} \sum_{C \in \mathcal{C}} \left(\sum_{v \in C} \deg(v) \right)^2$$

NP-hard to optimize [BDG⁺08]

Institute of Theoretical Informatics
Group Algorithmics

Modularity in Practice

- easy to use & implement
- reasonable behavior on many practical instances
  $\rightsquigarrow$ heavily used in various fields:
  - ecosystem exploration
  - collaboration analyses
  - biochemistry
  - structure of the internet (AS-graph, www, routers)
- close to human intuition of quality [GGHW10]

Modularity in Practice

- easy to use & implement
- reasonable behavior on many practical instances
  $\rightsquigarrow$ heavily used in various fields:
  - ecosystem exploration
  - collaboration analyses
  - biochemistry
  - structure of the internet (AS-graph, www, routers)
- close to human intuition of quality [GGHW10]

- scaling behavior (double instance, result differs) [folklore]
- non-locality of optimal clustering [folklore]
- resolution limit (no tiny and large clusters at the same time) [FB07]
- large sparse graph $\rightsquigarrow$ high values, balanced clusters [GdMC10]

Surprise

$G = (V, E)$, $|E| = m$, clustering $\mathcal{C}$, i_e intracluster edges
Random $\mathcal{G}$ with m edges

Surprise

$$S(\mathcal{C}) := \text{Prob}(\mathcal{G} \text{ has at least } i_e \text{ intracluster edges in } \mathcal{C}) = \sum_{i=i_e}^{m} \frac{\binom{i_p}{i} \cdot \binom{p-i_p}{m-i}}{\binom{p}{m}},$$

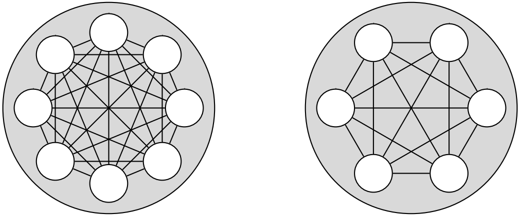
where $p := \binom{n}{2}$ and i_p #intra-cluster node pairs. [AMM05]

Urn model: i_p white, $p - i_p$ black balls, draw m balls w/o replacement

- NP-hard to optimize [FKW14]

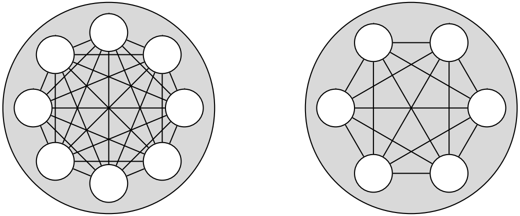
Cluster Editing

Ideal clustering: Disjoint cliques.



Cluster Editing

Ideal clustering: Disjoint cliques.

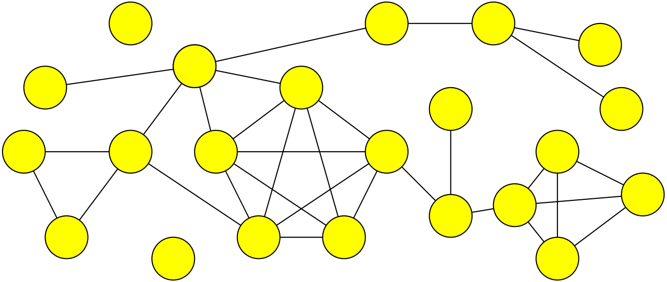


Idea: Edge editing to disjoint cliques – **Cluster Editing**.

Institute of Theoretical Informatics
Group Algorithmics

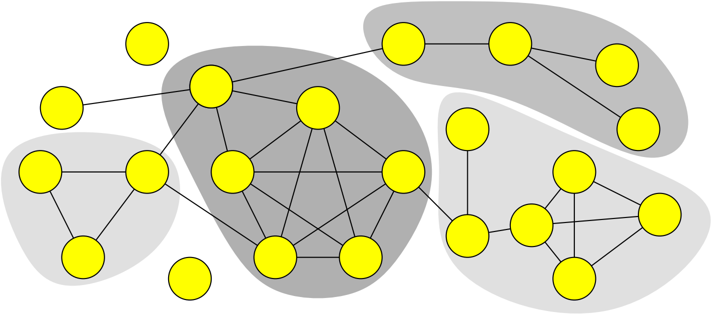
Cluster Editing

“How many edges must be inserted or deleted to arrive at disjoint cliques?”



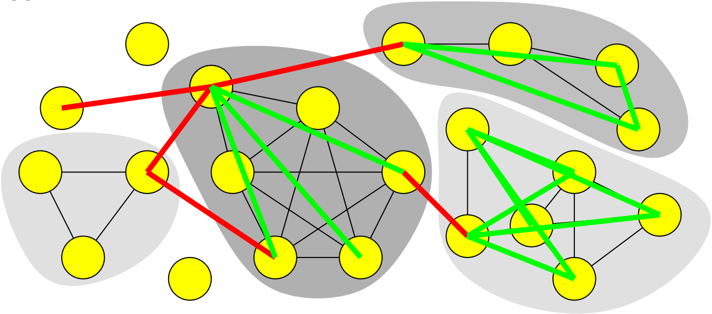
Cluster Editing

"How many edges must be inserted or deleted to arrive at disjoint cliques?"



Cluster Editing

"How many edges must be inserted or deleted to arrive at disjoint cliques?"

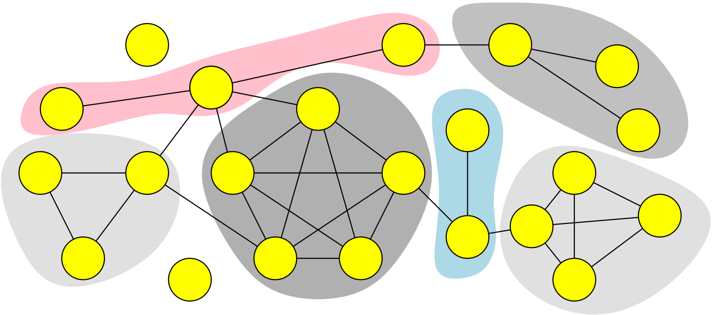


editing set has size 5 + 12 = 17 (bad)

Institute of Theoretical Informatics
Group Algorithmics

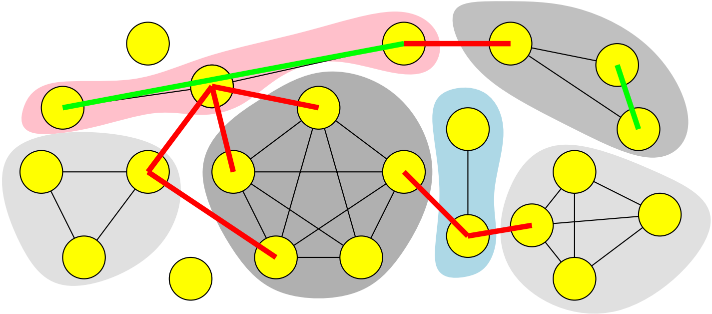
Cluster Editing

"How many edges must be inserted or deleted to arrive at disjoint cliques?"



Cluster Editing

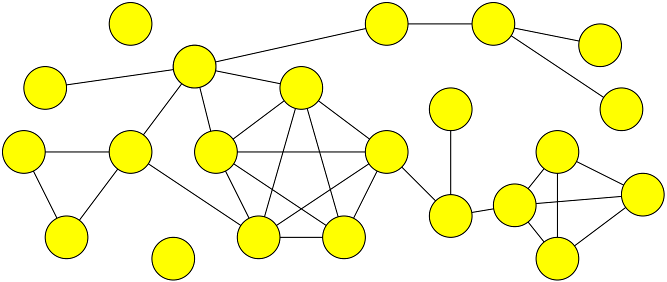
"How many edges must be inserted or deleted to arrive at disjoint cliques?"



editing set has size 7 + 3 = 10 (better)

Cluster Editing

"How many edges must be inserted or deleted to arrive at disjoint cliques?"



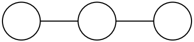
Task: find clustering with minimum cluster editing set [BB13, BBK08]

- NP-complete
- popular in biology

Institute of Theoretical Informatics
Group Algorithmics

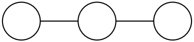
Forbidden Subgraphs

Disjoint cliques $\Leftrightarrow$ no P_3 as node-induced subgraph

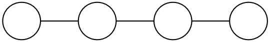


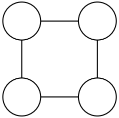
Forbidden Subgraphs

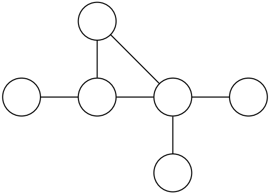
Disjoint cliques $\Leftrightarrow$ no P_3 as node-induced subgraph



Generalization:
No P_4 or C_4 as node-induced subgraph
= *Quasi-Threshold Graph*

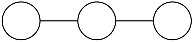




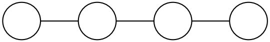


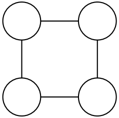
Forbidden Subgraphs

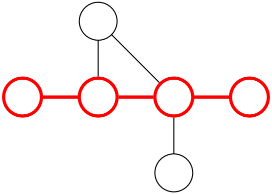
Disjoint cliques ⇔ no P_3 as node-induced subgraph



Generalization:
No P_4 or C_4 as node-induced subgraph
= *Quasi-Threshold Graph*



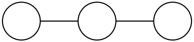




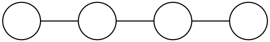
Institute of Theoretical Informatics
Group Algorithmics

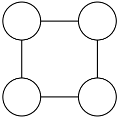
Forbidden Subgraphs

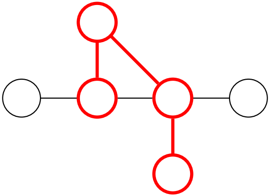
Disjoint cliques ⇔ no P_3 as node-induced subgraph



Generalization:
No P_4 or C_4 as node-induced subgraph
= *Quasi-Threshold Graph*

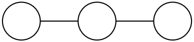




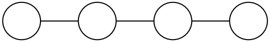


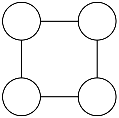
Forbidden Subgraphs

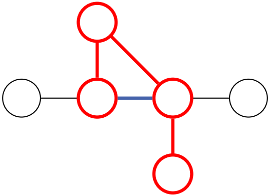
Disjoint cliques $\Leftrightarrow$ no P_3 as node-induced subgraph



Generalization:
No P_4 or C_4 as node-induced subgraph
= *Quasi-Threshold Graph*

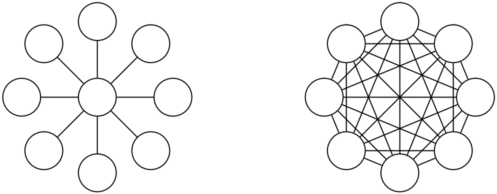






Quasi-Threshold Graphs

- Trivially perfect graphs
- Dense? Sparse? – Both!



- Max. diameter 2
- Central hub per component

Institute of Theoretical Informatics
Group Algorithmics

Why Quasi-Threshold Editing?

Karlsruhe Institute of Technology

- Components of quasi-threshold graphs are communities [NG13]

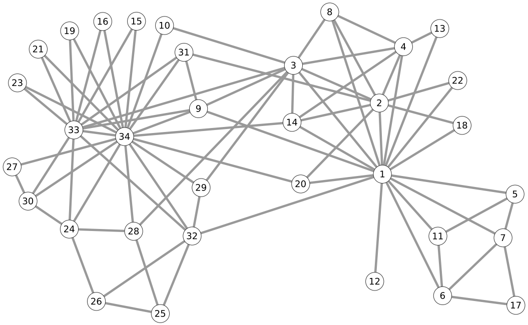
Institute of Theoretical Informatics
Group Algorithmics

Why Quasi-Threshold Editing?

- Components of quasi-threshold graphs are communities [NG13]
- Real world graphs are not quasi-threshold graphs
  $\rightsquigarrow$ Find quasi-threshold graph with small edge edit distance

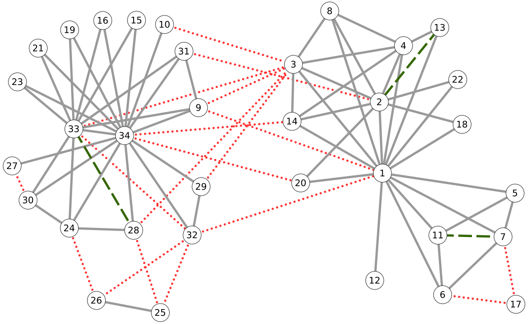
Why Quasi-Threshold Editing?

- Components of quasi-threshold graphs are communities [NG13]
- Real world graphs are not quasi-threshold graphs
  $\rightsquigarrow$ Find quasi-threshold graph with small edge edit distance



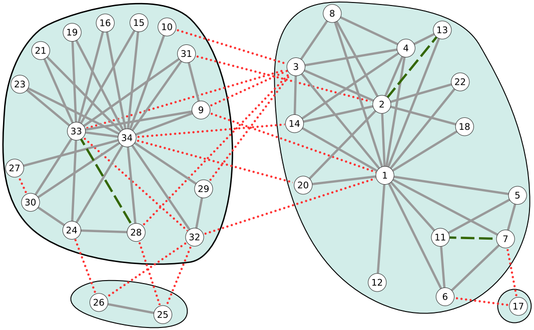
Why Quasi-Threshold Editing?

- Components of quasi-threshold graphs are communities [NG13]
- Real world graphs are not quasi-threshold graphs
  ⇝ Find quasi-threshold graph with small edge edit distance



Why Quasi-Threshold Editing?

- Components of quasi-threshold graphs are communities [NG13]
- Real world graphs are not quasi-threshold graphs
  $\rightsquigarrow$ Find quasi-threshold graph with small edge edit distance



Algorithmic Results

- General
- Cluster Editing
- Quasi-Threshold Editing
- Threshold Editing

Approaches

Karlsruhe Institute of Technology

P_3-free editing and P_4/C_4-free editing are NP-complete
$\Rightarrow$ no efficient exact algorithms in general

Alternative approaches:

- Average case instead of worst case analysis
- Randomization
- Approximative solutions
- Fixed parameter tractability (FPT)
- Empirical studying of heuristics on benchmarks

Approaches

P_3-free editing and P_4/C_4-free editing are NP-complete
$\Rightarrow$ no efficient exact algorithms in general

Alternative approaches:

- Average case instead of worst case analysis
- Randomization
- Approximative solutions
- Fixed parameter tractability (FPT)
- Empirical studying of heuristics on benchmarks

Fixed Parameter Tractability

Goal: Limit the explosion of the running time: $O(f(k) \cdot n^{O(1)})$.
Challenge: Identifying a suitable, “small” parameter k.

$\oplus$: Optimal, provable running time
$\ominus$: Exponential running time

Hope:

- Obtain a small kernel in polynomial time
- “Tolerable” $f(k)$

Institute of Theoretical Informatics
Group Algorithmics

Formal Definition

Parameterized Problem

$L \subseteq \Sigma^* \times \Sigma^*$ (usually $\subseteq \Sigma^* \times \mathbb{N}$)

Fixed-parameter tractable

$L \in \mathcal{FPT}$ iff $(x, k) \in L$ can be decided in time $f(k) \cdot |x|^{O(1)}$ where f is a computable function only depending on k.

Kernelization

$(x, k) \mapsto (x', k')$, with

- $k' \leq k$, $|x'| \leq g(k)$
- $(x, k) \in L$ iff $(x', k') \in L$
- Reduction in polynomial time

Institute of Theoretical Informatics
Group Algorithmics

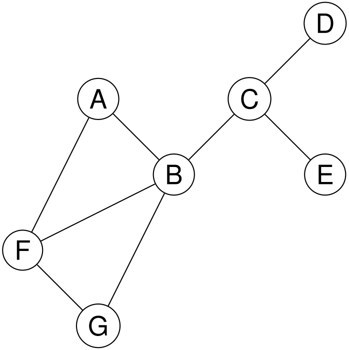
General Results

Graph classes defined by a finite set of (finite) forbidden induced subgraphs:

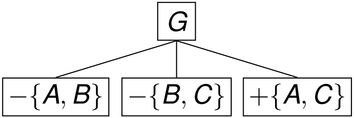
- Editing FPT in number of edits k, $O\left(\nu^{2k} \cdot n^{\nu+1}\right)$, ν maximum number of nodes in a forbidden subgraph. [Cai96]

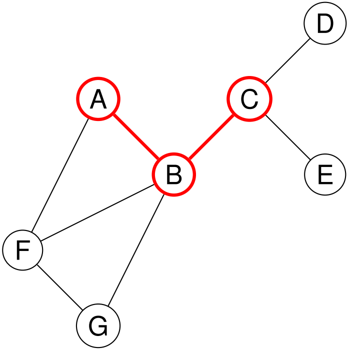
Example: P_3-free, $k = 3$

$\boxed{G}$



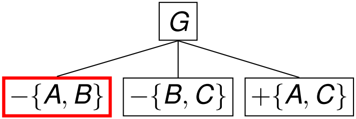
Example: P_3-free, $k = 3$

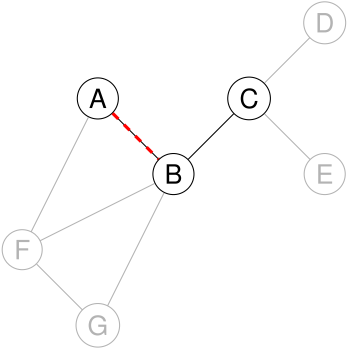




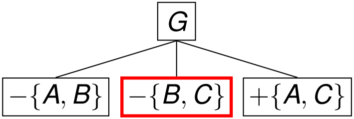
Institute of Theoretical Informatics
Group Algorithmics

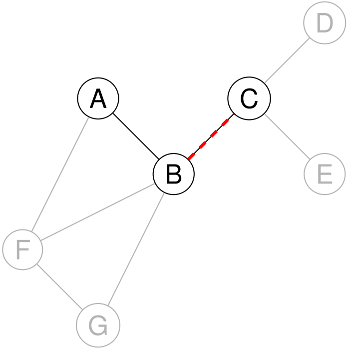
Example: P_3-free, $k = 3$



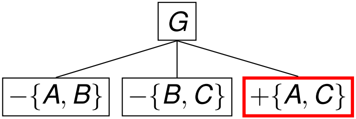


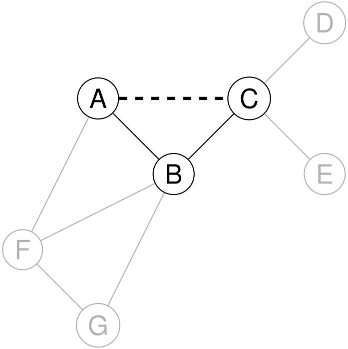
Example: P_3-free, $k = 3$



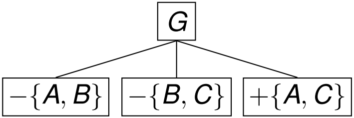


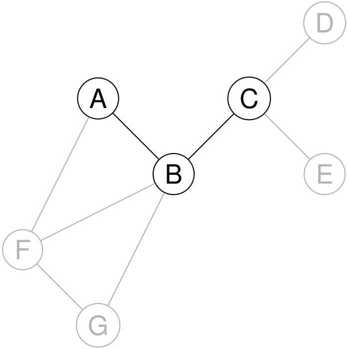
Example: P_3-free, $k = 3$



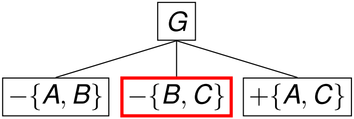


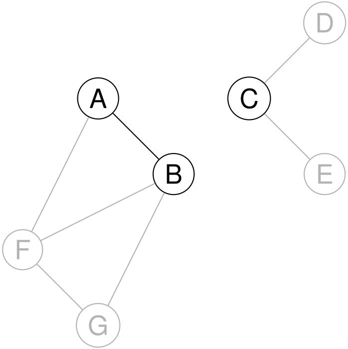
Example: P_3-free, $k = 3$



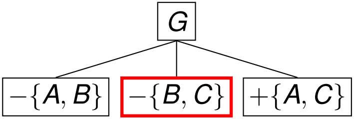


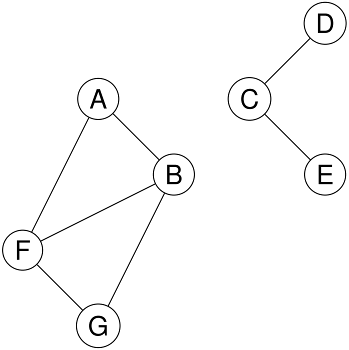
Example: P_3-free, $k = 3$



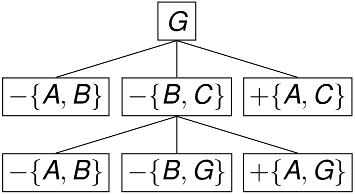


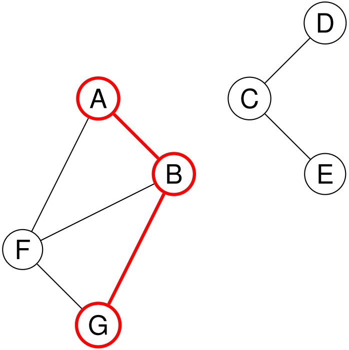
Example: P_3-free, $k = 3$





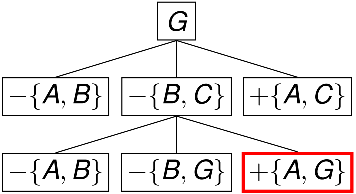
Example: P_3-free, $k = 3$

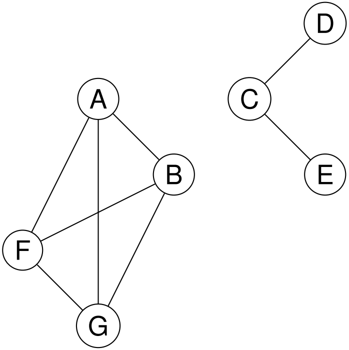




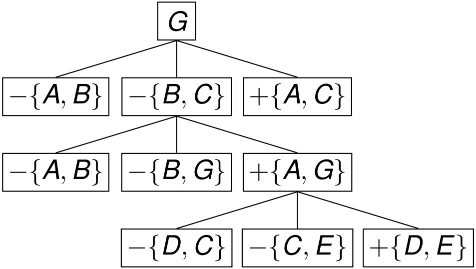
Institute of Theoretical Informatics
Group Algorithmics

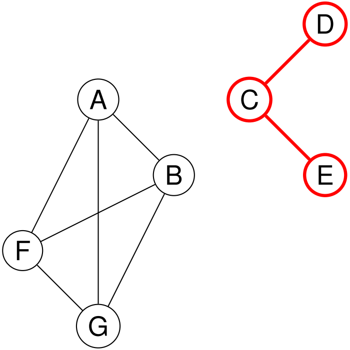
Example: P_3-free, $k = 3$





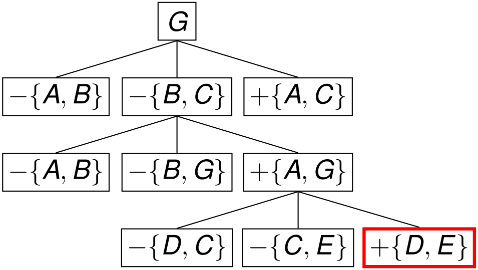
Example: P_3-free, $k = 3$

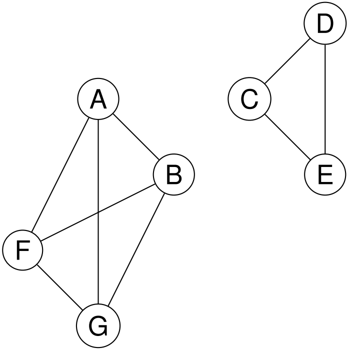




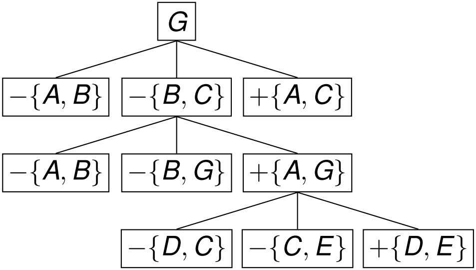
Institute of Theoretical Informatics
Group Algorithmics

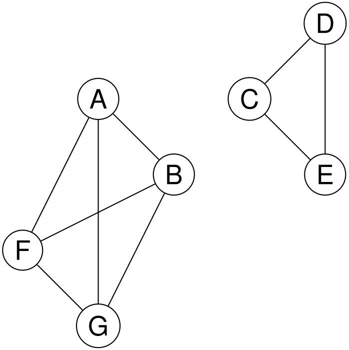
Example: P_3-free, $k = 3$





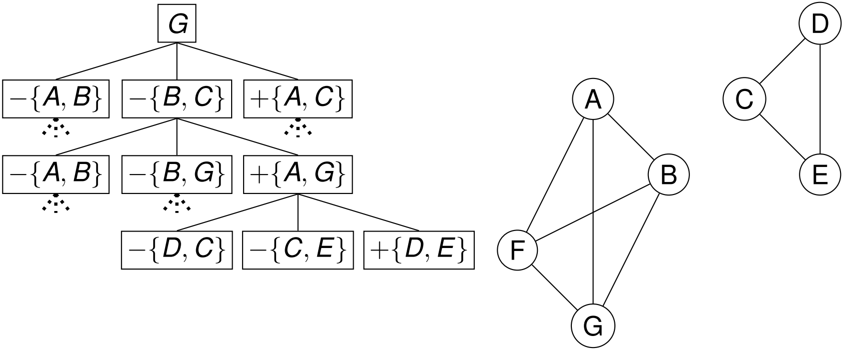
Example: P_3-free, $k = 3$





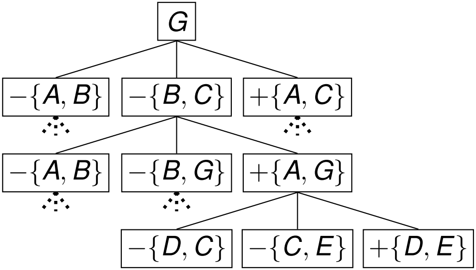
$\Rightarrow$ Found solution.

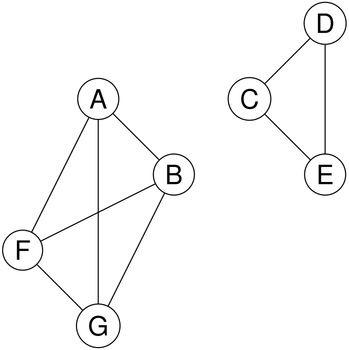
Example: P_3-free, $k = 3$



$\Rightarrow$ Found solution. If not: need to search the full tree.
If nothing found at level k: impossible with k edits.

Example: P_3-free, $k = 3$





$\Rightarrow$ Found solution. If not: need to search the full tree.
If nothing found at level k: impossible with k edits.

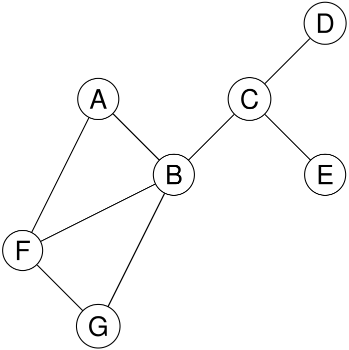
- Time $O(3^k \cdot \text{poly}(n))$
- Best known: $O(1.62^k + m + n)$ [Böc12]

FPT-based editing

- Show for a graph that k is exact solution:
  - show solution with k
  - show impossibility with $k-1$
- Branching rules can be optimized automatically [GGHN03]
- Bounding possible to limit explored branches.

Lower Bounds: P_3-free, $k = 3$

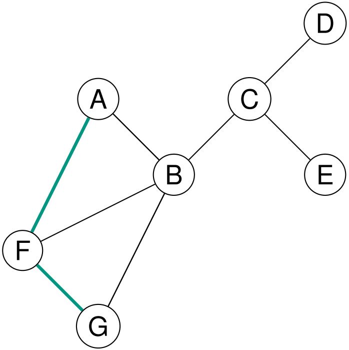
$\boxed{G}$



Lower Bound: 0
Remaining operations: 3

Lower Bounds: P_3-free, $k = 3$

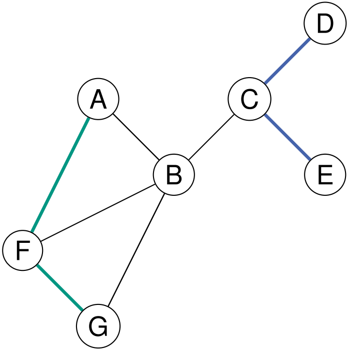
$\boxed{G}$



Lower Bound: 1
Remaining operations: 3

Lower Bounds: P_3-free, $k = 3$

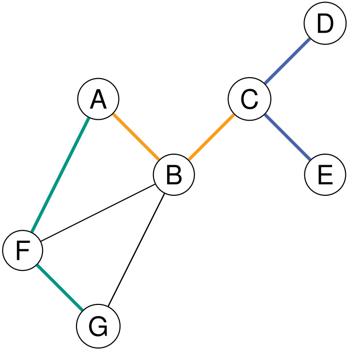
$\boxed{G}$



Lower Bound: 2
Remaining operations: 3

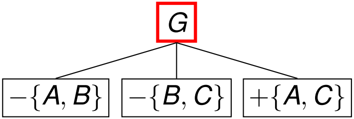
Lower Bounds: P_3-free, $k = 3$

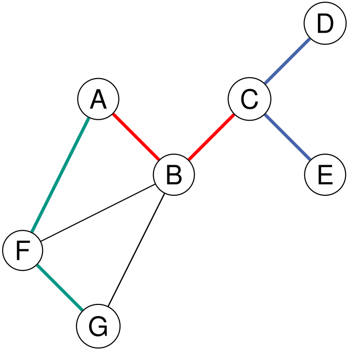
$\boxed{G}$



Lower Bound: 3
Remaining operations: 3

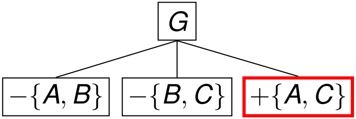
Lower Bounds: P_3-free, $k = 3$

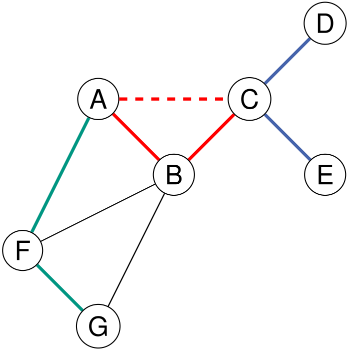




Lower Bound: 3
Remaining operations: 3

Lower Bounds: P_3-free, $k = 3$

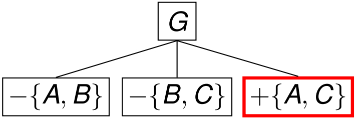


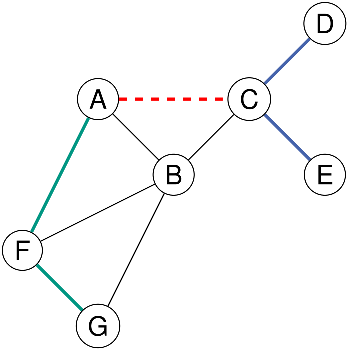


Lower Bound: 3
Remaining operations: 2

Institute of Theoretical Informatics
Group Algorithmics

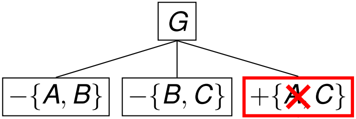
Lower Bounds: P_3-free, $k = 3$

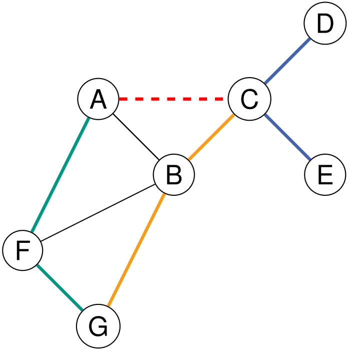




Lower Bound: 2
Remaining operations: 2

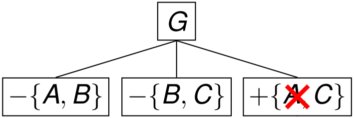
Lower Bounds: P_3-free, $k = 3$

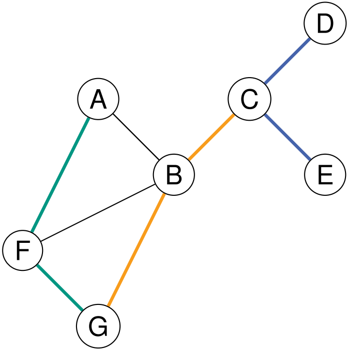




Lower Bound: 3
Remaining operations: 2

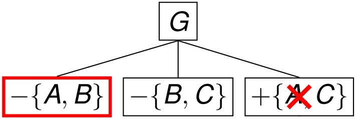
Lower Bounds: P_3-free, $k = 3$

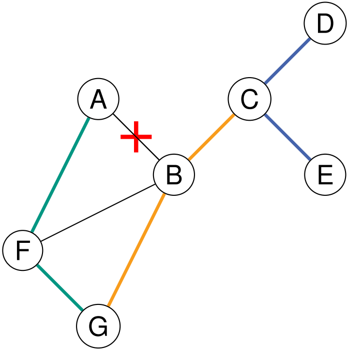




Lower Bound: 3
Remaining operations: 3

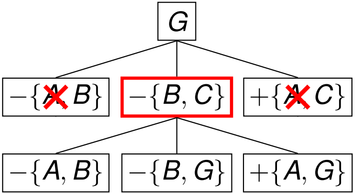
Lower Bounds: P_3-free, $k = 3$

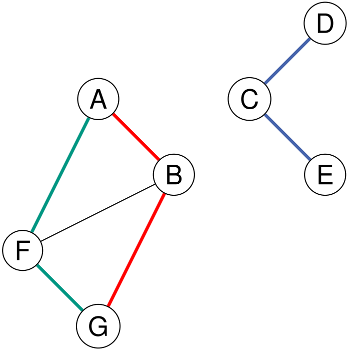




Lower Bound: 3
Remaining operations: 2

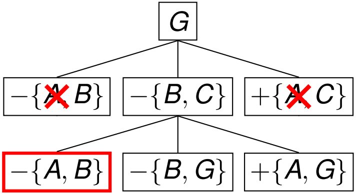
Lower Bounds: P_3-free, $k = 3$

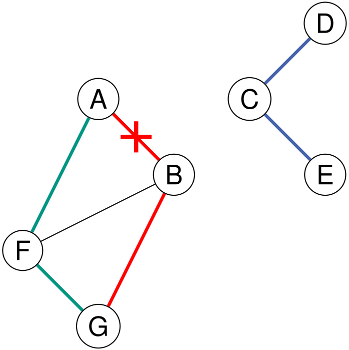




Lower Bound: 2
Remaining operations: 2

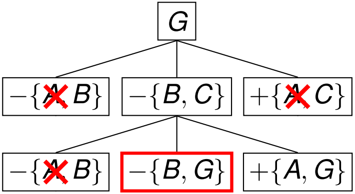
Lower Bounds: P_3-free, $k = 3$

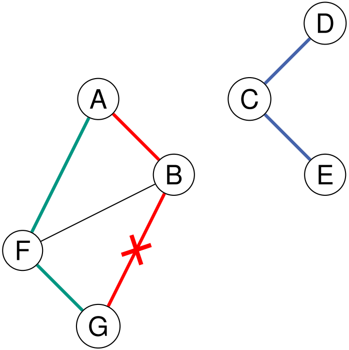




Lower Bound: 2
Remaining operations: 1

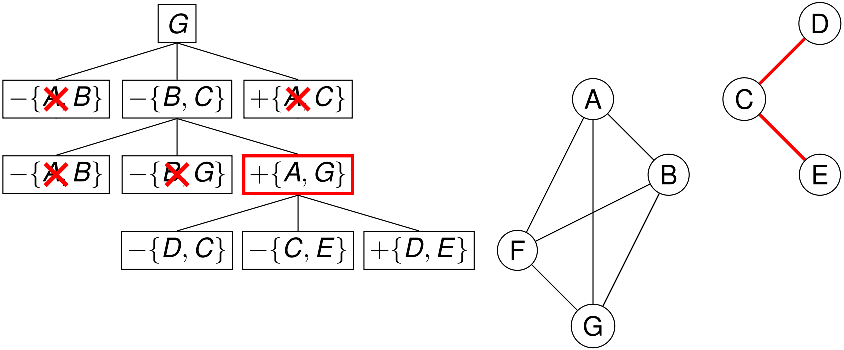
Lower Bounds: P_3-free, $k = 3$





Lower Bound: 2
Remaining operations: 1

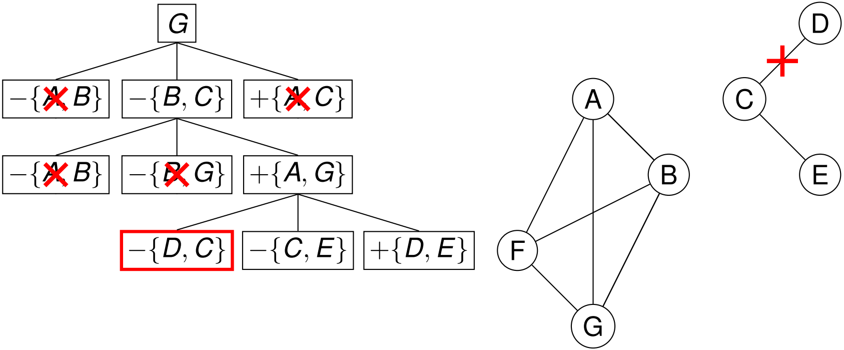
Lower Bounds: P_3-free, $k = 3$



Lower Bound: 1
Remaining operations: 1

Institute of Theoretical Informatics
Group Algorithmics

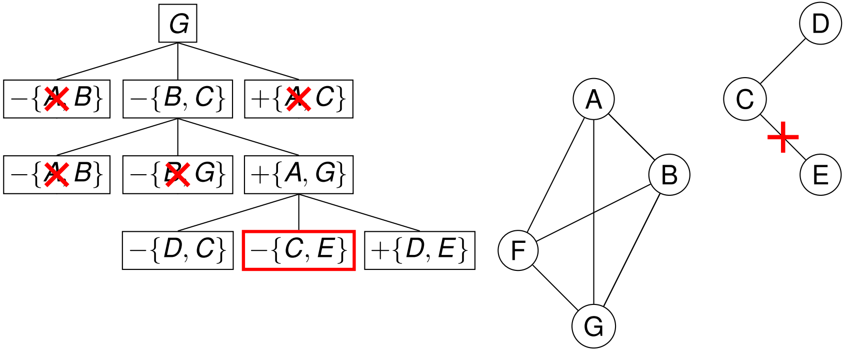
Lower Bounds: P_3-free, $k = 3$



Lower Bound: 0
Remaining operations: 0

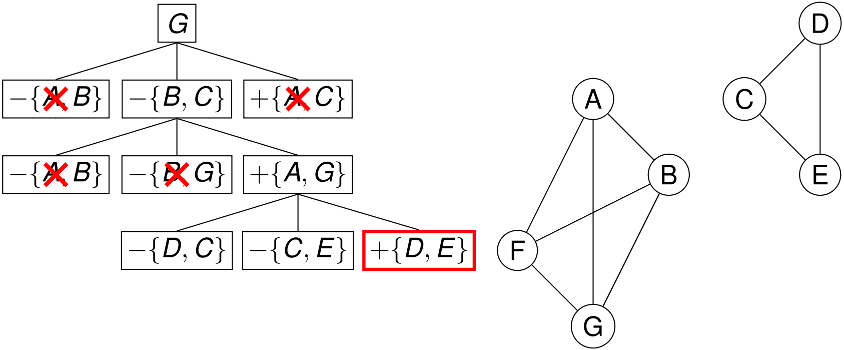
Institute of Theoretical Informatics
Group Algorithmics

Lower Bounds: P_3-free, $k = 3$



Lower Bound: 0
Remaining operations: 0

Lower Bounds: P_3-free, $k = 3$



Lower Bound: 0
Remaining operations: 0

Cluster Editing

- There is a $2k$-kernel ($k =$ number of edits) [CM12]
- Heuristics exist [BB13]

Quasi-Threshold Editing

Karlsruhe Institute of Technology

Quasi-Threshold Editing Problem

Given a graph G find a quasi-threshold graph with minimum edge editing (insertion + deletion) distance to G.

Quasi-Threshold Editing

Karlsruhe Institute of Technology

Quasi-Threshold Editing Problem

Given a graph G find a quasi-threshold graph with minimum edge editing (insertion + deletion) distance to G.

Quasi-Threshold Recognition:

- Certifying recognition in linear time. [Chu08]
- Simpler certifying recognition algorithm [BHSW15]

Quasi-Threshold Editing

Quasi-Threshold Editing Problem

Given a graph G find a quasi-threshold graph with minimum edge editing (insertion + deletion) distance to G.

Quasi-Threshold Recognition:

- Certifying recognition in linear time. [Chu08]
- Simpler certifying recognition algorithm [BHSW15]

Exact editing:

- Is NP-hard [NG13]
- Is FPT $O\left(6^k\left(|V|+|E|\right)\right)$ [Cai96]
- Polynomial kernel exists ($O\left(k^7\right)$ vertices) [DP15]

Quasi-Threshold Editing

Quasi-Threshold Editing Problem

Given a graph G find a quasi-threshold graph with minimum edge editing (insertion + deletion) distance to G.

Quasi-Threshold Recognition:

- Certifying recognition in linear time. [Chu08]
- Simpler certifying recognition algorithm [BHSW15]

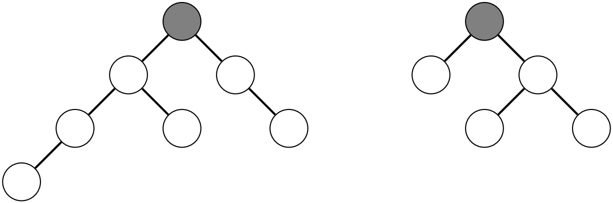
Exact editing:

- Is NP-hard [NG13]
- Is FPT $O\left(6^k\left(|V|+|E|\right)\right)$ [Cai96]
- Polynomial kernel exists ($O\left(k^7\right)$ vertices) [DP15]

Heuristic editing:

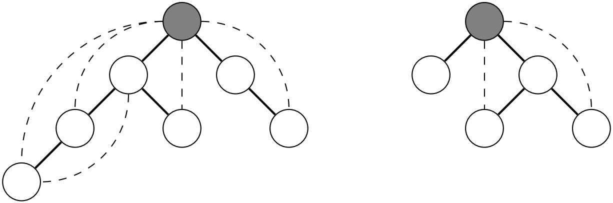
- First editing heuristic – $\Omega\left(|V|^2\right)$ [NG13]
- Faster editing heuristic: Quasi-Threshold Mover (QTM) [BHSW15]

Skeleton Forests

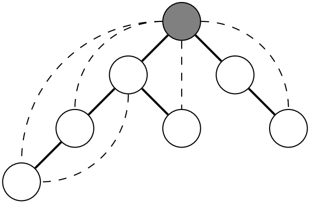


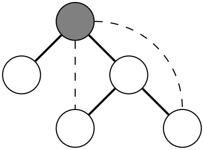
Institute of Theoretical Informatics
Group Algorithmics

Skeleton Forests



Skeleton Forests





Quasi-Threshold Graphs

Quasi-threshold graphs are exactly the transitive closure of rooted forests.

Certifying Algorithms

Certifying Algorithm

A certifying algorithm is an algorithm that produces, with each output, a certificate or witness (easy-to-verify proof) that the particular output has not been compromised by a bug. [MMNS11]

Quasi-Threshold Recognition:

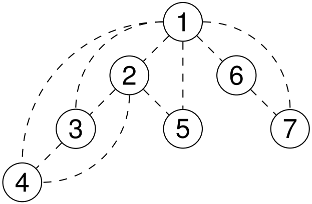
- Positive proof: A skeleton forest such that the graph is its transitive closure.
- Negative proof: An induced P_4 or C_4.

Quasi-Threshold Recognition

$p(u)$ parent of node u, init with -1;
foreach *$u \in V$ sorted by degree in decreasing order* **do**
    foreach *non-processed neighbor c of u* **do**
        if $p(u) = p(c)$ **then**
            Set $p(c)$ to u;
        else
            Construct P_4 or C_4;
    Mark u as processed;

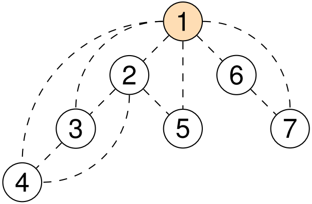
Quasi-Threshold Recognition

$p(u)$ parent of node u, init with -1;
foreach *$u \in V$ sorted by degree in decreasing order* **do**
    foreach *non-processed neighbor c of u* **do**
        if $p(u) = p(c)$ **then**
            Set $p(c)$ to u;
        else
            Construct P_4 or C_4;
    Mark u as processed;



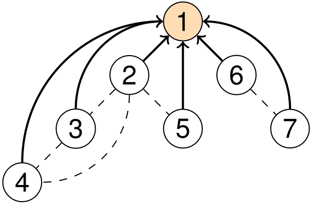
Quasi-Threshold Recognition

```
p(u) parent of node u, init with −1;
foreach u ∈ V sorted by degree in decreasing order do
    foreach non-processed neighbor c of u do
        if p(u) = p(c) then
            Set p(c) to u;
        else
            Construct P_4 or C_4;
    Mark u as processed;
```



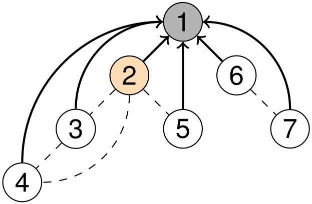
Quasi-Threshold Recognition

```
p(u) parent of node u, init with −1;
foreach u ∈ V sorted by degree in decreasing order do
    foreach non-processed neighbor c of u do
        if p(u) = p(c) then
            Set p(c) to u;
        else
            Construct P4 or C4;
    Mark u as processed;
```



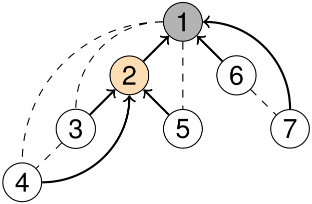
Quasi-Threshold Recognition

```
p(u) parent of node u, init with −1;
foreach u ∈ V sorted by degree in decreasing order do
    foreach non-processed neighbor c of u do
        if p(u) = p(c) then
            Set p(c) to u;
        else
            Construct P_4 or C_4;
    Mark u as processed;
```



Quasi-Threshold Recognition

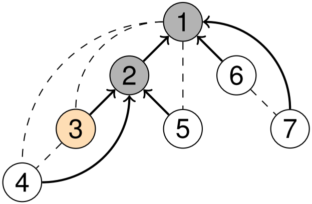
```
p(u) parent of node u, init with −1;
foreach u ∈ V sorted by degree in decreasing order do
    foreach non-processed neighbor c of u do
        if p(u) = p(c) then
            Set p(c) to u;
        else
            Construct P_4 or C_4;
    Mark u as processed;
```



Quasi-Threshold Recognition

Karlsruhe Institute of Technology

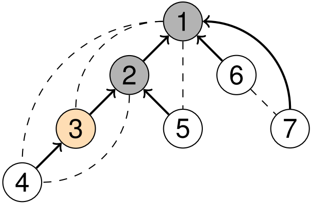
```
p(u) parent of node u, init with −1;
foreach u ∈ V sorted by degree in decreasing order do
    foreach non-processed neighbor c of u do
        if p(u) = p(c) then
            Set p(c) to u;
        else
            Construct P4 or C4;
    Mark u as processed;
```



Institute of Theoretical Informatics
Group Algorithmics

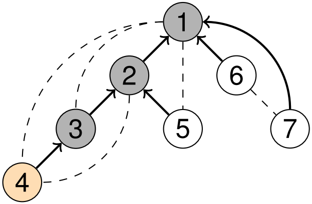
Quasi-Threshold Recognition

```
p(u) parent of node u, init with −1;
foreach u ∈ V sorted by degree in decreasing order do
    foreach non-processed neighbor c of u do
        if p(u) = p(c) then
            Set p(c) to u;
        else
            Construct P_4 or C_4;
    Mark u as processed;
```



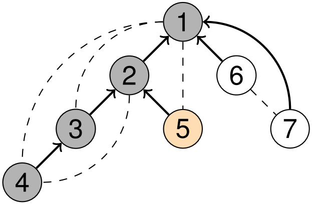
Quasi-Threshold Recognition

$p(u)$ parent of node u, init with -1;
foreach *$u \in V$ sorted by degree in decreasing order* **do**
    foreach *non-processed neighbor c of u* **do**
        if $p(u) = p(c)$ **then**
            Set $p(c)$ to u;
        else
            Construct P_4 or C_4;
    Mark u as processed;



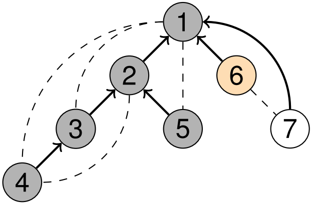
Quasi-Threshold Recognition

$p(u)$ parent of node u, init with -1;
foreach *$u \in V$ sorted by degree in decreasing order* **do**
  foreach *non-processed neighbor c of u* **do**
    if $p(u) = p(c)$ **then**
      Set $p(c)$ to u;
    else
      Construct P_4 or C_4;
  Mark u as processed;



Quasi-Threshold Recognition

```
p(u) parent of node u, init with −1;
foreach u ∈ V sorted by degree in decreasing order do
    foreach non-processed neighbor c of u do
        if p(u) = p(c) then
            Set p(c) to u;
        else
            Construct P4 or C4;
    Mark u as processed;
```

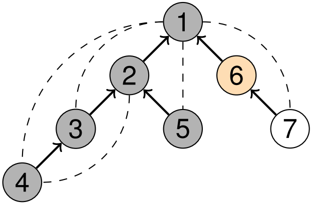


Institute of Theoretical Informatics
Group Algorithmics

Quasi-Threshold Recognition

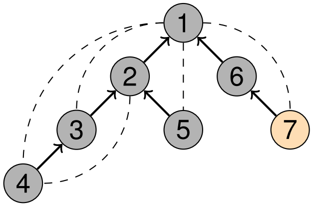
Karlsruhe Institute of Technology

```
p(u) parent of node u, init with −1;
foreach u ∈ V sorted by degree in decreasing order do
    foreach non-processed neighbor c of u do
        if p(u) = p(c) then
            Set p(c) to u;
        else
            Construct P4 or C4;
    Mark u as processed;
```



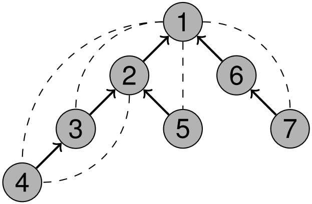
Quasi-Threshold Recognition

```
p(u) parent of node u, init with −1;
foreach u ∈ V sorted by degree in decreasing order do
    foreach non-processed neighbor c of u do
        if p(u) = p(c) then
            Set p(c) to u;
        else
            Construct P4 or C4;
    Mark u as processed;
```



Quasi-Threshold Recognition

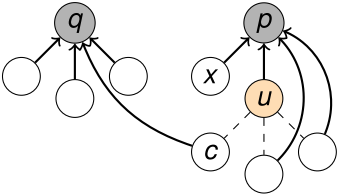
$p(u)$ parent of node u, init with -1;
foreach *$u \in V$ sorted by degree in decreasing order* **do**
    foreach *non-processed neighbor c of u* **do**
        if $p(u) = p(c)$ **then**
            Set $p(c)$ to u;
        else
            Construct P_4 or C_4;
    Mark u as processed;



Institute of Theoretical Informatics
Group Algorithmics

Quasi-Threshold Recognition

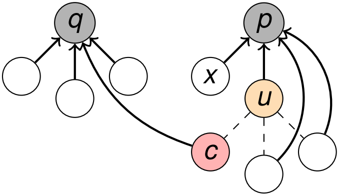
```
p(u) parent of node u, init with −1;
foreach u ∈ V sorted by degree in decreasing order do
    foreach non-processed neighbor c of u do
        if p(u) = p(c) then
            Set p(c) to u;
        else
            Construct P_4 or C_4;
    Mark u as processed;
```



Quasi-Threshold Recognition

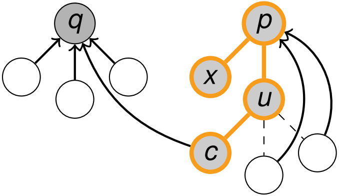
Karlsruhe Institute of Technology

```
p(u) parent of node u, init with −1;
foreach u ∈ V sorted by degree in decreasing order do
    foreach non-processed neighbor c of u do
        if p(u) = p(c) then
            Set p(c) to u;
        else
            Construct P4 or C4;
    Mark u as processed;
```

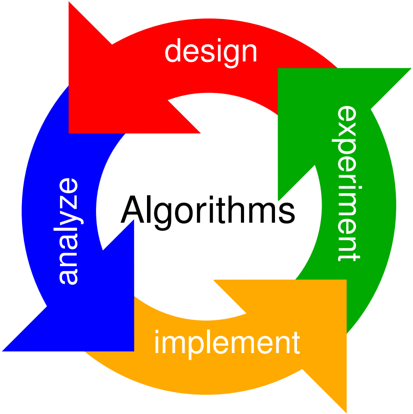


Quasi-Threshold Recognition

$p(u)$ parent of node u, init with -1;
foreach *$u \in V$ sorted by degree in decreasing order* **do**
    foreach *non-processed neighbor c of u* **do**
        if $p(u) = p(c)$ **then**
            Set $p(c)$ to u;
        else
            Construct P_4 or C_4;
    Mark u as processed;



Algorithm Engineering



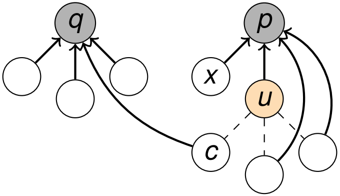
Initial Editing Heuristic

- Use recognition
- Resolve errors locally

Institute of Theoretical Informatics
Group Algorithmics

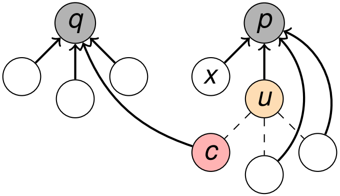
Initial Editing Heuristic

- Use recognition
- Resolve errors locally



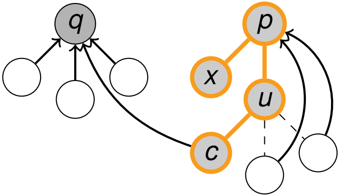
Initial Editing Heuristic

- Use recognition
- Resolve errors locally



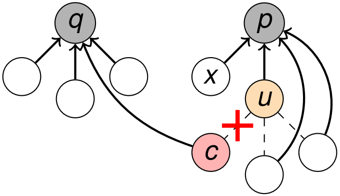
Initial Editing Heuristic

- Use recognition
- Resolve errors locally



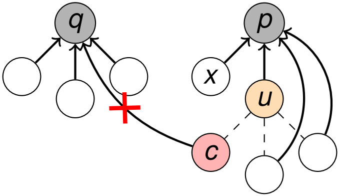
Initial Editing Heuristic

- Use recognition
- Resolve errors locally



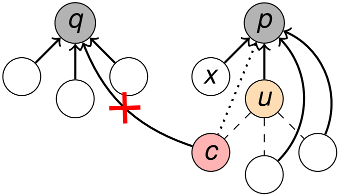
Initial Editing Heuristic

- Use recognition
- Resolve errors locally



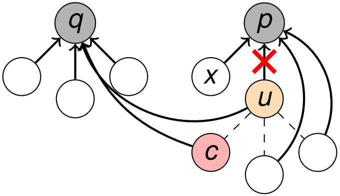
Initial Editing Heuristic

- Use recognition
- Resolve errors locally



Initial Editing Heuristic

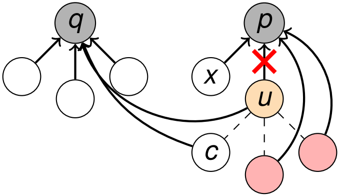
- Use recognition
- Resolve errors locally



Institute of Theoretical Informatics
Group Algorithmics

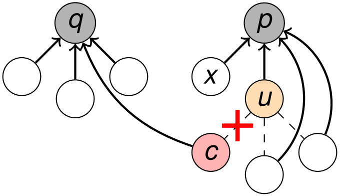
Initial Editing Heuristic

- Use recognition
- Resolve errors locally



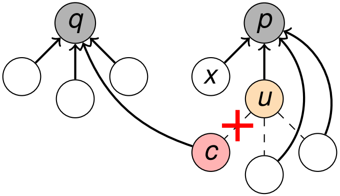
Initial Editing Heuristic

- Use recognition
- Resolve errors locally



Initial Editing Heuristic

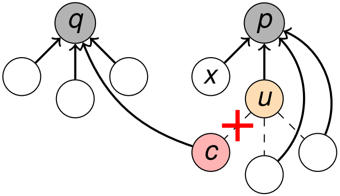
- Use recognition
- Resolve errors locally



- Use triangles and depth for decisions
  $\rightsquigarrow$ High number of edits but yields good initialization

Initial Editing Heuristic

- Use recognition
- Resolve errors locally



- Use triangles and depth for decisions
  $\rightsquigarrow$ High number of edits but yields good initialization
- Time: Triangle counting $O(\alpha \cdot |E|)$ + linear

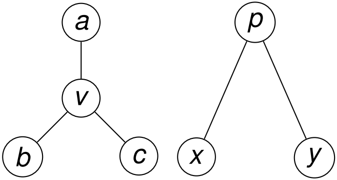
QTM: Basic Idea

Modify skeleton forest using local moving

For each node apply move:

- Choose parent
- Adopt children

such that #edits is minimum among choices.



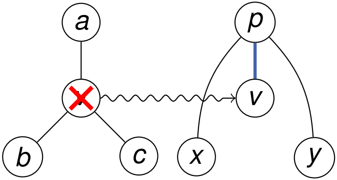
QTM: Basic Idea

Modify skeleton forest using local moving

For each node apply move:

- Choose parent
- Adopt children

such that #edits is minimum among choices.



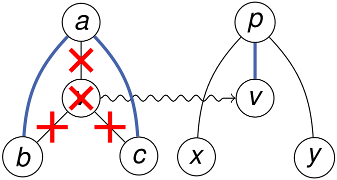
QTM: Basic Idea

Modify skeleton forest using local moving

For each node apply move:

- Choose parent
- Adopt children

such that #edits is minimum among choices.



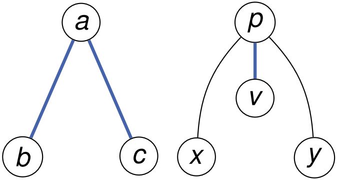
QTM: Basic Idea

Modify skeleton forest using local moving

For each node apply move:

- Choose parent
- Adopt children

such that #edits is minimum among choices.



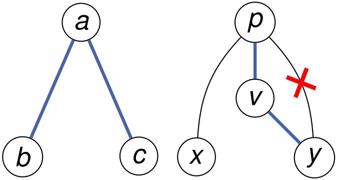
QTM: Basic Idea

Modify skeleton forest using local moving

For each node apply move:

- Choose parent
- Adopt children

such that #edits is minimum among choices.



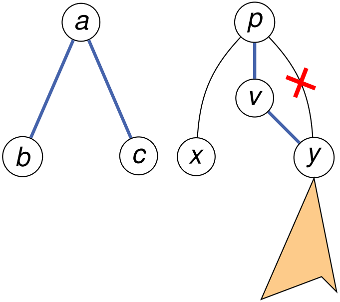
QTM: Basic Idea

Modify skeleton forest using local moving

For each node apply move:

- Choose parent
- Adopt children

such that #edits is minimum among choices.



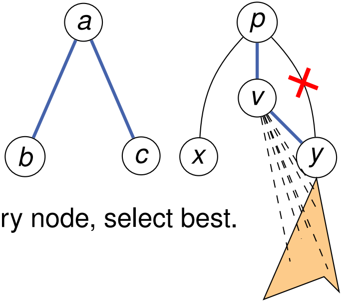
QTM: Basic Idea

Modify skeleton forest using local moving

For each node apply move:

- Choose parent
- Adopt children

such that #edits is minimum among choices.



Count (non-)neighbors below and above every node, select best.

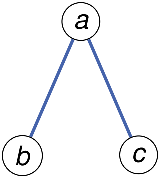
QTM: Basic Idea

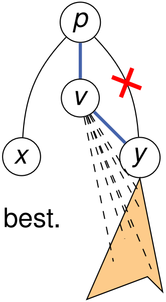
Modify skeleton forest using local moving

For each node apply move:

- Choose parent
- Adopt children

such that #edits is minimum among choices.





Count (non-)neighbors below and above every node, select best.

Simple idea:

- Count (non-)neighbors below and above each node using a (single) DFS
- Select best node

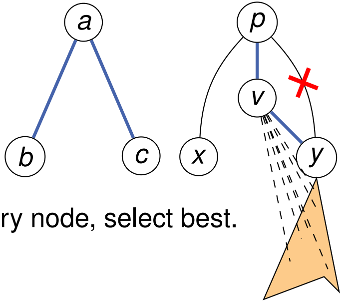
QTM: Basic Idea

Modify skeleton forest using local moving

For each node apply move:

- Choose parent
- Adopt children

such that #edits is minimum among choices.



Count (non-)neighbors below and above every node, select best.

Simple idea:

- Count (non-)neighbors below and above each node using a (single) DFS
- Select best node

Problem: time $O(|V|)$ per node, $O(|V|^2)$ per round.

Institute of Theoretical Informatics
Group Algorithmics

QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted

QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

Institute of Theoretical Informatics
Group Algorithmics

QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

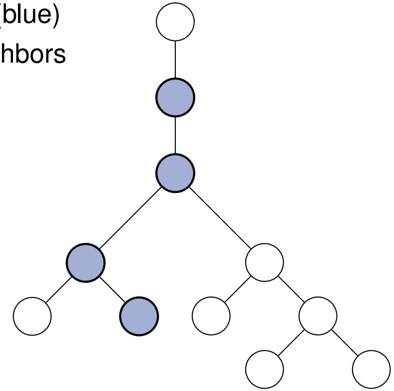
- Start at neighbors of node to move
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

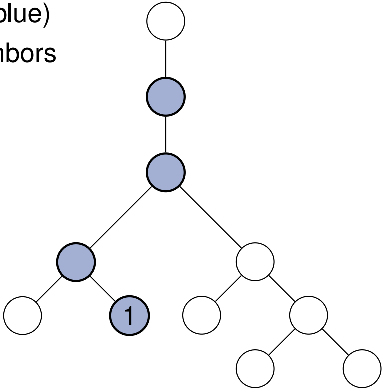


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

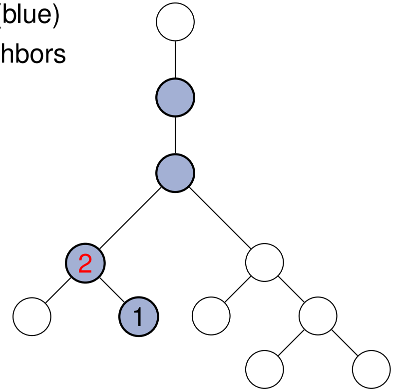


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

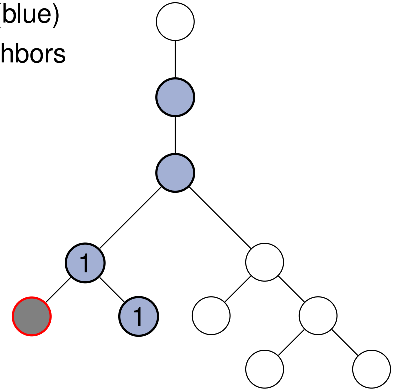


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

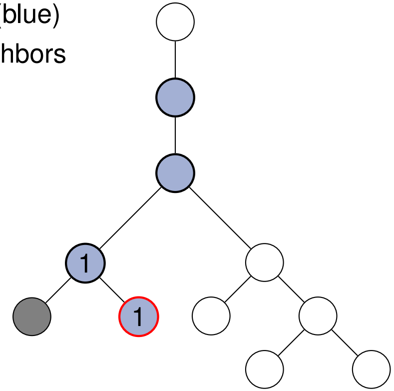


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

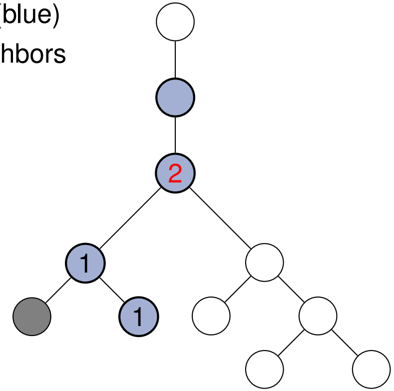


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor



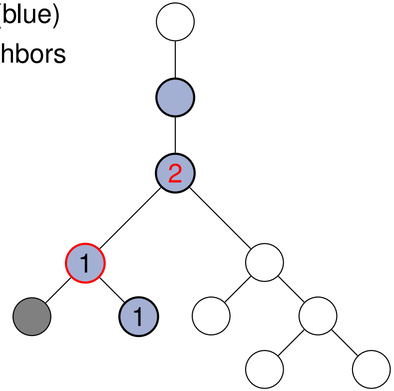
Institute of Theoretical Informatics
Group Algorithmics

QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

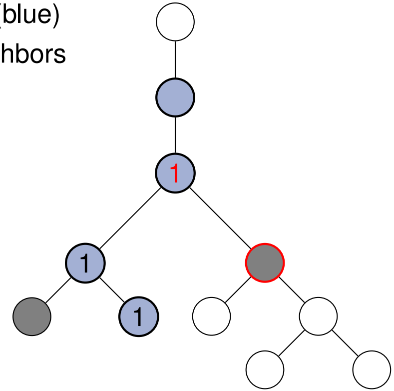


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

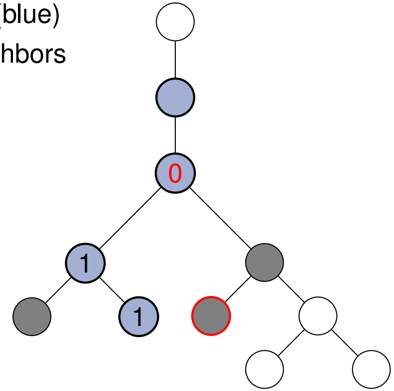


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

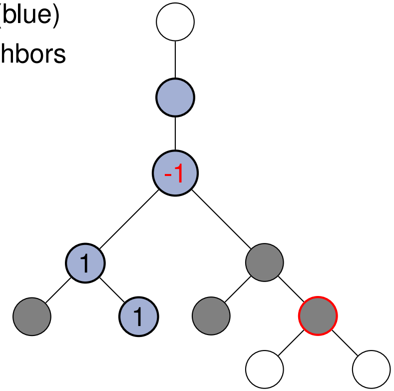


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

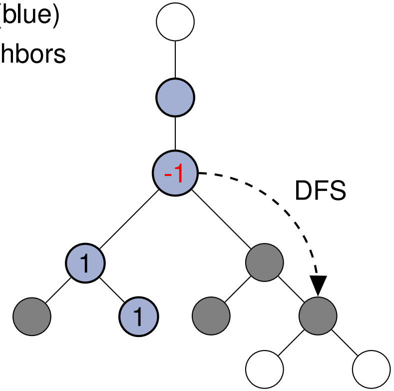


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

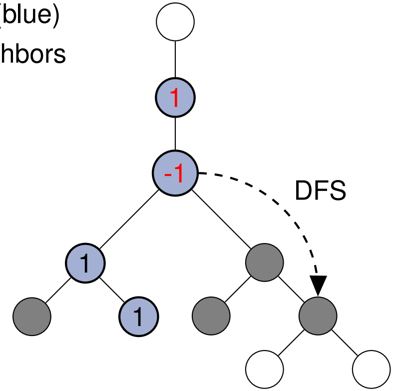


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

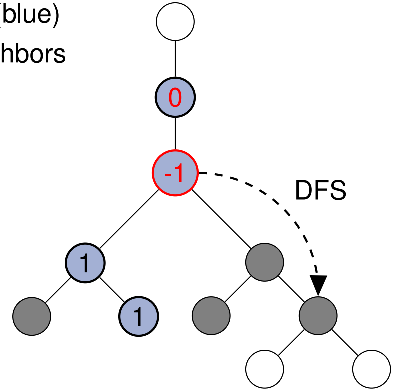


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

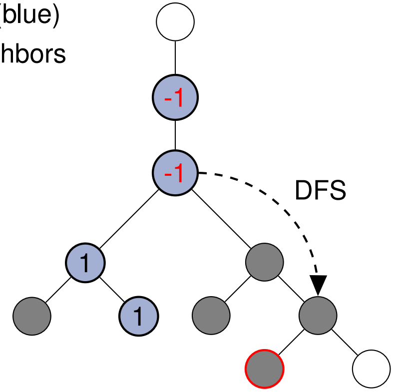


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

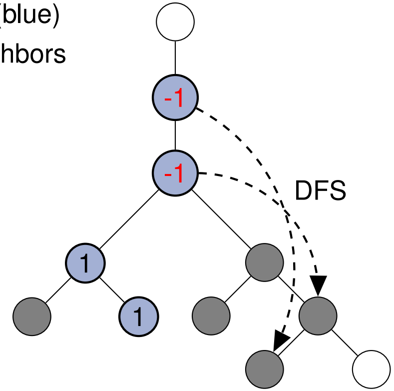


QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor



QTM: Fast Local Moving

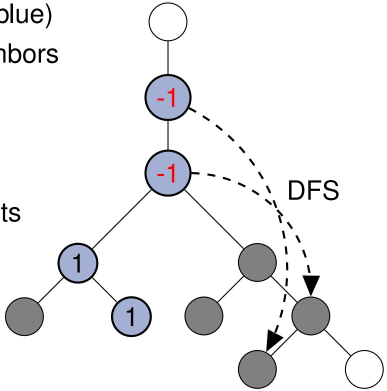
Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\rightsquigarrow$ visit $O(1)$ nodes per neighbor

Find best parent:

- Bottom-up scan from potential parents



QTM: Fast Local Moving

Parents: neighbors and nodes with children that should be adopted
Adopt children that have more neighbors than non-neighbors

How to evaluate children:

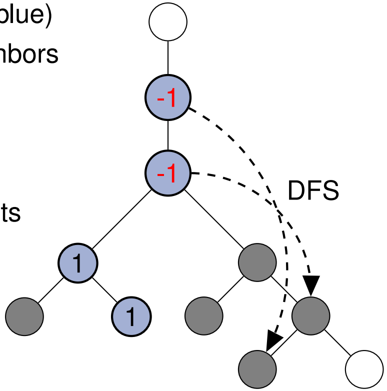
- Start at neighbors of node to move (blue)
- Bottom-up scan with surplus of neighbors
- Limited DFS when surplus exists
  $\leadsto$ visit $O(1)$ nodes per neighbor

Find best parent:

- Bottom-up scan from potential parents

Time:

- Amortized $O(d \log(d))$ per node
- $O(|E| \log d_{\max})$ per round
- 4 rounds enough in practice



Evaluation: Comparison to previous heuristic

On large networks previous heuristic too slow.

QTM:

Name	$\|V\|$ [K]	$\|E\|$ [K]	Edits [K]	Time [s]
Caltech [TMP12]	0.77	16.66	11.6	<0.1
Orkut [LK14]	3 072	117 185	103 426	866.4
uk-2002 [BV04]	18 520	261 787	31 218	1 638.0

Evaluation: Synthetic networks

Generation:

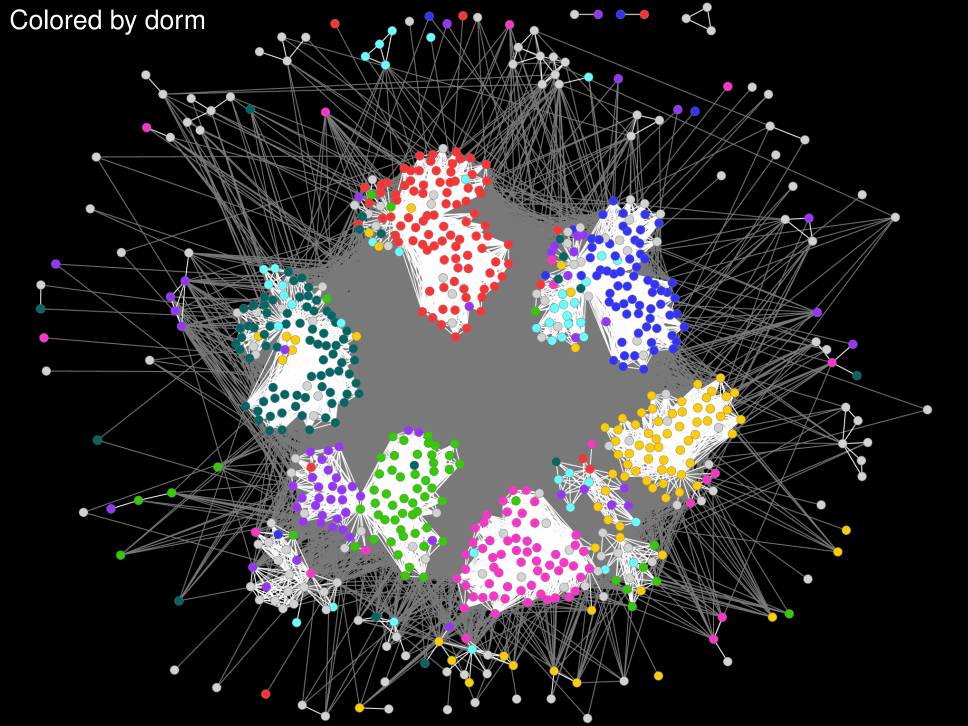
- Generate quasi-threshold graphs
- Introduce edit difference by random edge deletions and insertions

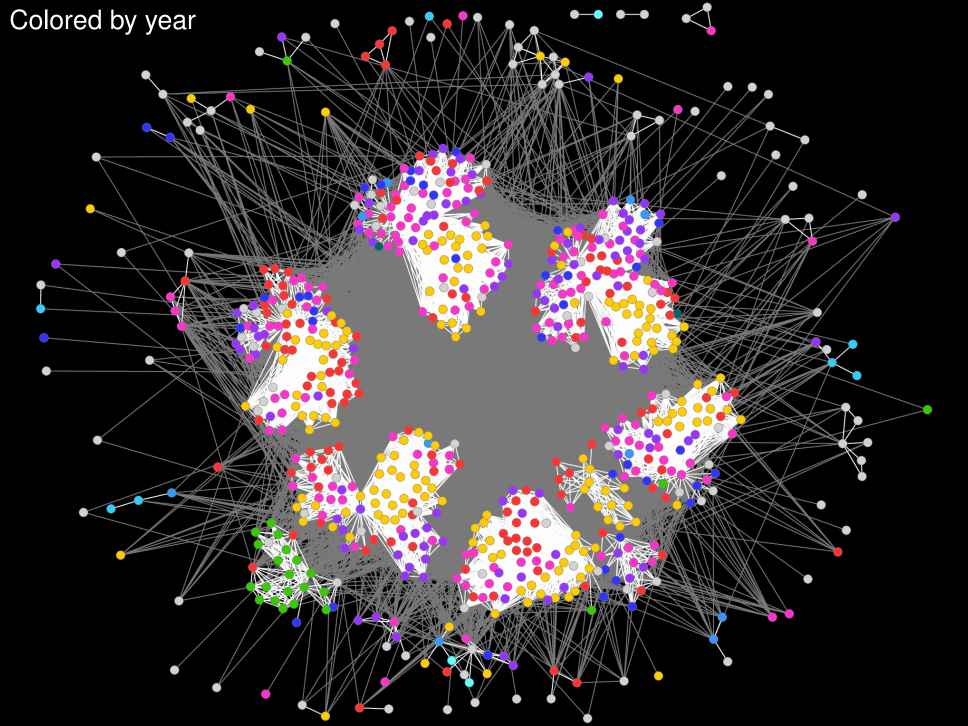
Result:

- QTM results as close or closer than generated quasi-threshold graphs

Case study: Caltech network

- Caltech Facebook network from September 2005 [TMP12]
- Nodes: 769 university members (mostly students)
- Edges: friendship on Facebook
- Anonymized node attributes:
  - Dormitory
  - Class year
  - Gender
  - Major
  - High school
- Dormitory, year correlated with edges


Colored by dorm


Colored by year

Conclusion

- Many problems can be formulated using graph editing
- Clustering – different formalizations using edge editing
- Both exact (FPT) and heuristic algorithms available

Conclusion

Karlsruhe Institute of Technology

- Many problems can be formulated using graph editing
- Clustering – different formalizations using edge editing
- Both exact (FPT) and heuristic algorithms available

Outlook

- Many more possible editing problems, e.g. P_5, C_5 – no good heuristics known
- Other variants using core-periphery structure even allow overlapping communities [BHK15]

Thank you!

Bibliography I

Karlsruhe Institute of Technology

Vicente Arnau, Sergio Mars, and Ignacio Marín.
Iterative Cluster Analysis of Protein Interaction Data.
Bioinformatics, 21(3):364–378, 2005.

Sebastian Böcker and Jan Baumbach.
Cluster Editing.
In *Proceedings of the 9th Conference on Computability in Europe (CiE'13)*, volume 7921 of *Lecture Notes in Computer Science*, pages 33–44. Springer, 2013.

Sebastian Böcker, Sebastian Briesemeister, and Gunnar W. Klau.
Exact Algorithms for Cluster Editing: Evaluation and Experiments.
In Catherine C. McGeoch, editor, *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 289–302. Springer, June 2008.

Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Höfer, Zoran Nikoloski, and Dorothea Wagner.
On Modularity Clustering.
IEEE Transactions on Knowledge and Data Engineering, 20(2):172–188, February 2008.

Sharon Bruckner, Falk Hüffner, and Christian Komusiewicz.
A graph modification approach for finding core-periphery structures in protein interaction networks.
Algorithms for Molecular Biology, 10, 2015.

Ulrik Brandes, Michael Hamann, Ben Strasser, and Dorothea Wagner.
Fast Quasi-Threshold Editing.
In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA'15)*, Lecture Notes in Computer Science. Springer, 2015.

Bibliography II

Sebastian Böcker.
A golden ratio parameterized algorithm for Cluster Editing.
Journal of Discrete Algorithms, 16:79–89, October 2012.

Paolo Boldi and Sebastiano Vigna.
The WebGraph Framework I: Compression Techniques.
In *Proceedings of the 13th International Conference on World Wide Web (WWW2004)*, pages 595–602. ACM Press, 2004.

Leizhen Cai.
Fixed-parameter tractability of graph modification problems for hereditary properties.
Information Processing Letters, 58(4):171–176, May 1996.

Frank Pok Man Chu.
A simple linear time certifying LBFS-based algorithm for recognizing trivially perfect graphs and their complements.
Information Processing Letters, 107(1):7–12, June 2008.

Jianer Chen and Jie Meng.
A 2k kernel for the cluster editing problem.
Journal of Computer and System Sciences, 78(1):211–220, January 2012.

Pål Grønås Drange and Michał Pilipczuk.
A Polynomial Kernel for Trivially Perfect Editing.
In *Proceedings of the 23rd Annual European Symposium on Algorithms (ESA'15)*, Lecture Notes in Computer Science. Springer, 2015.

Santo Fortunato and Marc Barthélemy.
Resolution limit in community detection.
Proceedings of the National Academy of Science of the United States of America, 104(1):36–41, 2007.

Bibliography III

Tobias Fleck, Andrea Kappes, and Dorothea Wagner.
Graph Clustering with Surprise: Complexity and Exact Solutions.
In *Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'14)*, volume 8327 of *Lecture Notes in Computer Science*, pages 223–234. Springer, 2014.

Benjamin H. Good, Yves-Alexandre de Montjoye, and Aaron Clauset.
Performance of modularity maximization in practical contexts.
Physical Review E, 81(046106), 2010.

Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier.
Automated Generation of Search Tree Algorithms for Graph Modification Problems.
In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA'03)*, volume 2832 of *Lecture Notes in Computer Science*, pages 642–653. Springer, 2003.

Robert Görke, Marco Gaertler, Florian Hübner, and Dorothea Wagner.
Computational Aspects of Lucidity-Driven Graph Clustering.
Journal of Graph Algorithms and Applications, 14(2):165–197, 2010.

Jon M. Kleinberg.
An Impossibility Theorem for Clustering.
In *Proceedings of 15th Conference: Neural Information Processing Systems*, 2002.

Jure Leskovec and Andrej Krevl.
SNAP Datasets: Stanford Large Network Dataset Collection, June 2014.

Ross McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer.
Certifying algorithms.
Computer Science Review, 5(2):119–161, May 2011.

Bibliography IV

Mark E. J. Newman and Michelle Girvan.
Finding and evaluating community structure in networks.
Physical Review E, 69(026113):1–16, 2004.

James Nastos and Yong Gao.
Familial groups in social networks.
Social Networks, 35(3):439–450, July 2013.

Amanda L. Traud, Peter J. Mucha, and Mason A. Porter.
Social structure of Facebook networks.
Physica A: Statistical Mechanics and its Applications, 391(16):4165–4180, August 2012.