# Fixed-Parameter Linear-Time Algorithms for Graph and Hypergraph Problems Arising in Industrial Applications

René van Bevern

Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany

Wissenschaftliche Aussprache, 17. Juni 2014

# NP-hard problems considered in the thesis

## *d*-Hitting Set

- Race condition detection in parallel Java programs.
  [O'Callahan and Choi, ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 2003]

## Dag Partitioning

- Real-time tracking of trends and topics on the internet.
  [Leskovec, Backstrom, and Kleinberg, ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2009]
  [Suen, Huang, Eksombatchai, Sosic, and Leskovec, International Conference on World Wide Web 2013]

## 2-Union Independent Set

- Job scheduling, e. g. in steel manufacturing.
  [Höhn, König, Möhring, and Lübbecke, Management Science 57, 2011]

## Hypergraph Cutwidth

- Automatic testing of circuits.
  [Prasad, Chong, and Keutzer, Design Automation Conference 1999]

# Fixed-parameter algorithms

**Challenge:** No polynomial-time algorithms for **NP-hard problems** known.

**Goal:** Solve NP-hard problems efficiently if certain **parameters** of the input are **small**.

# Fixed-parameter algorithms

**Challenge:** No polynomial-time algorithms for **NP-hard problems** known.

**Goal:** Solve NP-hard problems efficiently if certain **parameters** of the input are **small**.

**Approach:** Solve problems in $f(k) \cdot \mathrm{poly}(n)$ time for some parameter $k \rightsquigarrow$ **fixed-parameter algorithms**.

# Fixed-parameter algorithms

**Challenge:** No polynomial-time algorithms for **NP-hard problems** known.

**Goal:** Solve NP-hard problems efficiently if certain **parameters** of the input are **small**.

**Approach:** Solve problems in $f(k) \cdot \mathrm{poly}(n)$ time for some parameter $k \rightsquigarrow$ **fixed-parameter algorithms**.

**Common lines of research:**

- Make $f(k)$ smaller, e. g. improve it from $k^k$ to $2^k$.
- Find fixed-parameter algorithms for smaller parameters.

# Fixed-parameter algorithms

**Challenge:** No polynomial-time algorithms for **NP-hard problems** known.

**Goal:** Solve NP-hard problems efficiently if certain **parameters** of the input are **small**.

**Approach:** Solve problems in $f(k) \cdot \mathrm{poly}(n)$ time for some parameter $k \rightsquigarrow$ **fixed-parameter algorithms**.

**Common lines of research:**

- Make $f(k)$ smaller, e. g. improve it from $k^k$ to $2^k$.
- Find fixed-parameter algorithms for smaller parameters.

**Still:** Algorithms running in $O(2^k \cdot n^6)$ time.

# Fixed-parameter algorithms

**Challenge:** No polynomial-time algorithms for **NP-hard problems** known.

**Goal:** Solve NP-hard problems efficiently if certain **parameters** of the input are **small**.

**Approach:** Solve problems in $f(k) \cdot \text{poly}(n)$ time for some parameter $k \rightsquigarrow$ **fixed-parameter algorithms**.

**Common lines of research:**

- Make $f(k)$ smaller, e. g. improve it from $k^k$ to $2^k$.
- Find fixed-parameter algorithms for smaller parameters.

**Still:** Algorithms running in $O(2^k \cdot n^6)$ time.

**Focus change:** Solve problems in **linear time** for constant parameter values $\rightsquigarrow$ **fixed-parameter linear-time algorithms**.

# Fixed-parameter algorithms

**Challenge:** No polynomial-time algorithms for **NP-hard problems** known.

**Goal:** Solve NP-hard problems efficiently if certain **parameters** of the input are **small**.

**Approach:** Solve problems in $f(k) \cdot \mathrm{poly}(n)$ time for some parameter $k \rightsquigarrow$ **fixed-parameter algorithms**.

**Common lines of research:**

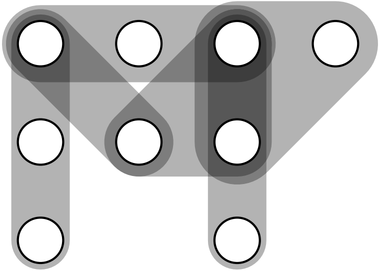
- Make $f(k)$ smaller, e. g. improve it from $k^k$ to $2^k$.
- Find fixed-parameter algorithms for smaller parameters.

**Still:** Algorithms running in $O(2^k \cdot n^6)$ time.

**Focus change:** Solve problems in **linear time** for constant parameter values $\rightsquigarrow$ **fixed-parameter linear-time algorithms**.

Back to the roots:
[Bodlaender, SIAM Journal on Computing 25, 1996:
"A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth"]

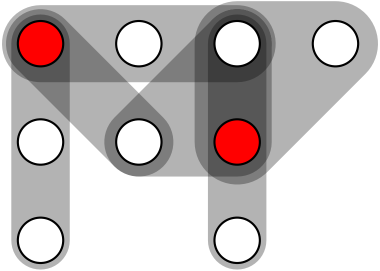
# $d$-Hitting Set



**Input:** A **hypergraph** $H = (V, E)$ with a set $V$ of **vertices**, a set $E \subseteq 2^V$ of **hyperedges**, each of cardinality at most a **constant $d$**, and a **natural number $k$**.

**Question:** Is there a **hitting set** $S \subseteq V$ of **size at most $k$**, that is, $\forall e \in E : S \cap E \neq \emptyset$?

# $d$-Hitting Set

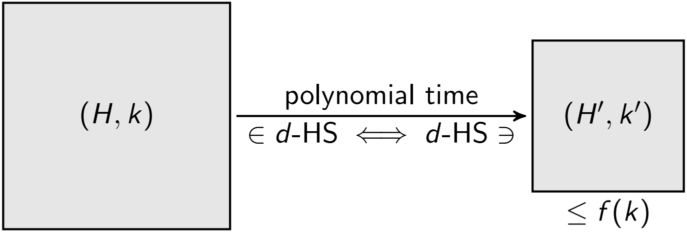


**Input:** A **hypergraph** $H = (V, E)$ with a set $V$ of **vertices**, a set $E \subseteq 2^V$ of **hyperedges**, each of cardinality at most a **constant** $\boldsymbol{d}$, and a **natural number** $\boldsymbol{k}$.

**Question:** Is there a **hitting set** $S \subseteq V$ of **size at most** $\boldsymbol{k}$, that is, $\forall e \in E : S \cap E \neq \emptyset$?

# Problem kernelization

$d$-HS is NP-hard $\rightsquigarrow$ data reduction, **problem kernelization**:



# Problem kernelization

$d$-HS is NP-hard $\rightsquigarrow$ data reduction, **problem kernelization**:

$$(H, k) \xrightarrow[\in d\text{-HS} \iff d\text{-HS} \ni]{\text{polynomial time}} (H', k')$$

$$\leq \cancel{f(k)}\ \mathrm{poly}(k)$$

# Problem kernelization

$d$-HS is NP-hard $\rightsquigarrow$ data reduction, **problem kernelization**:

$(H, k)$ $\xrightarrow[\in d\text{-HS} \iff d\text{-HS} \ni]{\cancel{\text{polynomial}}\text{ linear time}}$ $(H', k')$

$\leq \cancel{f(k)}\ \mathrm{poly}(k)$

# $d$-HS problem kernels

No $O(k^{d-\varepsilon})$-size problem kernel.

[Dell and van Melkebeek, ACM Symposium on Theory of Computing 2010]

$O(k^d)$-size problem kernels for $d$-HS in polynomial time.

[Flum and Grohe, Parameterized Complexity Theory, 2006]

[Damaschke, Theoretical Computer Science 351, 2006]

[S. Kratsch, Algorithmica 62, 2012]

$O(k^{d-1})$-vertex problem kernels for $d$-HS in polynomial time.

[Abu-Khzam, Journal of Computer and System Sciences 76, 2010]

[Moser, Dissertation, Friedrich-Schiller-Universität Jena, 2010]

# $d$-HS problem kernels

No $O(k^{d-\varepsilon})$-size problem kernel.

[Dell and van Melkebeek, ACM Symposium on Theory of Computing 2010]

$O(k^d)$-size problem kernels for $d$-HS in polynomial time.

[Flum and Grohe, Parameterized Complexity Theory, 2006]

[Damaschke, Theoretical Computer Science 351, 2006]

[S. Kratsch, Algorithmica 62, 2012]

$O(k^{d-1})$-vertex problem kernels for $d$-HS in polynomial time.

[Abu-Khzam, Journal of Computer and System Sciences 76, 2010]

[Moser, Dissertation, Friedrich-Schiller-Universität Jena, 2010]

**New:** $O(k^d)$-size problem kernel in $O(|V| + |E|)$ time.

# $d$-HS problem kernels

No $O(k^{d-\varepsilon})$-size problem kernel.

[Dell and van Melkebeek, ACM Symposium on Theory of Computing 2010]

$O(k^d)$-size problem kernels for $d$-HS in polynomial time.

[Flum and Grohe, Parameterized Complexity Theory, 2006]

[Damaschke, Theoretical Computer Science 351, 2006]

[S. Kratsch, Algorithmica 62, 2012]

$O(k^{d-1})$-vertex problem kernels for $d$-HS in polynomial time.

[Abu-Khzam, Journal of Computer and System Sciences 76, 2010]

[Moser, Dissertation, Friedrich-Schiller-Universität Jena, 2010]

**New:** $O(k^d)$-size problem kernel in $O(|V| + |E|)$ time.

- Running time and size are essentially optimal.

# $d$-HS problem kernels

No $O(k^{d-\varepsilon})$-size problem kernel.

[Dell and van Melkebeek, ACM Symposium on Theory of Computing 2010]

$O(k^d)$-size problem kernels for $d$-HS in polynomial time.

[Flum and Grohe, Parameterized Complexity Theory, 2006]

[Damaschke, Theoretical Computer Science 351, 2006]

[S. Kratsch, Algorithmica 62, 2012]

$O(k^{d-1})$-vertex problem kernels for $d$-HS in polynomial time.

[Abu-Khzam, Journal of Computer and System Sciences 76, 2010]

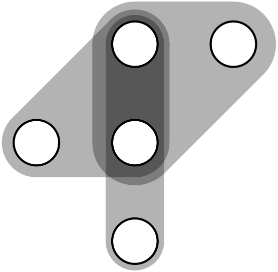
[Moser, Dissertation, Friedrich-Schiller-Universität Jena, 2010]

**New:** $O(k^d)$-size problem kernel in $O(|V| + |E|)$ time.

- Running time and size are essentially optimal.
- $d$-HS solvable in $O(d^k + |V| + |E|)$ time.

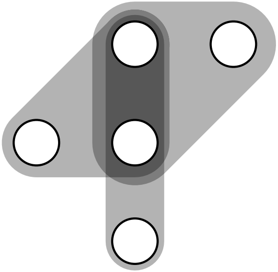
# Sunflowers

A family $P$ of sets that pairwise only intersect in a **core** $C$ is a **sunflower**. The sets in $P$ are **petals**.



# Sunflowers

A family $P$ of sets that pairwise only intersect in a **core** $C$ is a **sunflower**. The sets in $P$ are **petals**.



**Theorem:** If a $d$-uniform hypergraph $H$ has more than $\boldsymbol{k^d \cdot d!}$ **hyperedges**, then it contains a **sunflower of size $\boldsymbol{k}$** that can be found in **polynomial time**.

[Erdős and Rado, Journal of the London Mathematical Society 35(1), 1960]

[Flum and Grohe, Parameterized Complexity Theory, 2006]

# Pruning sunflowers

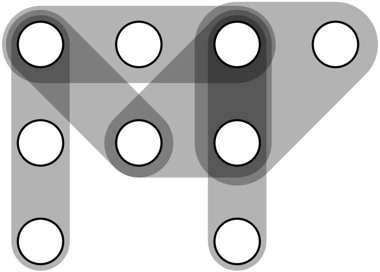
Given a $d$-HS instance $(H, k)$, **repeatedly**

- try to **find** a sunflower with $k + 2$ petals,
- **remove** one of its petals from $H$. [S. Kratsch, Algorithmica 63, 2012]

# Pruning sunflowers

Given a $d$-HS instance $(H, k)$, **repeatedly**

- try to **find** a sunflower with $k + 2$ petals,
- **remove** one of its petals from $H$. [S. Kratsch, Algorithmica 63, 2012]

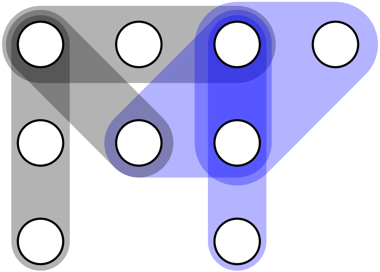


We try to kernelize this instance for $k = 1$.

# Pruning sunflowers

Given a $d$-HS instance $(H, k)$, **repeatedly**

- try to **find** a sunflower with $k + 2$ petals,
- **remove** one of its petals from $H$. [S. Kratsch, Algorithmica 63, 2012]

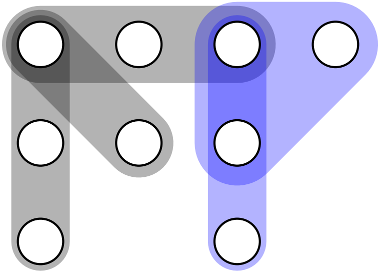


Find sunflower of size $k + 2 = 3$.

# Pruning sunflowers

Given a $d$-HS instance $(H, k)$, **repeatedly**

- try to **find** a sunflower with $k + 2$ petals,
- **remove** one of its petals from $H$. [S. Kratsch, Algorithmica 63, 2012]

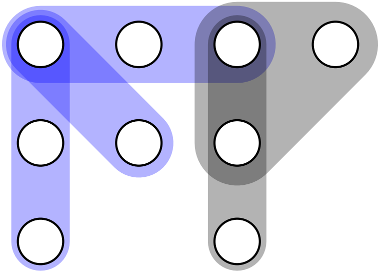


Remove one petal.

# Pruning sunflowers

Given a $d$-HS instance $(H, k)$, **repeatedly**

- try to **find** a sunflower with $k + 2$ petals,
- **remove** one of its petals from $H$. [S. Kratsch, Algorithmica 63, 2012]

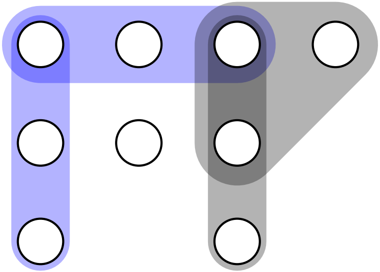


Find sunflower of size $k + 2 = 3$.

# Pruning sunflowers

Given a $d$-HS instance $(H, k)$, **repeatedly**

- try to **find** a sunflower with $k + 2$ petals,
- **remove** one of its petals from $H$. [S. Kratsch, Algorithmica 63, 2012]

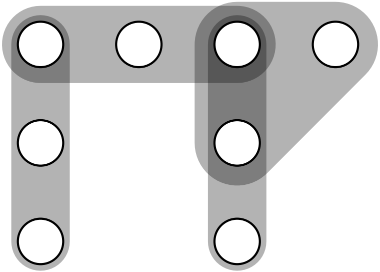


Remove one petal.

# Pruning sunflowers

Given a $d$-HS instance $(H, k)$, **repeatedly**

- try to **find** a sunflower with $k + 2$ petals,
- **remove** one of its petals from $H$. [S. Kratsch, Algorithmica 63, 2012]



Resulting problem kernel.

# Growing sunflowers

Given a $d$-HS instance $(H, k)$, create an empty hypergraph $H'$.
Then, **for each** hyperedge $e$ of $H$:

- **if** $e$ does not contain the core of a $(k+1)$-petal sunflower in $H'$
  - **add** $e$ to $H'$.

Want this in constant time.

# Growing sunflowers

Given a $d$-HS instance $(H, k)$, create an empty hypergraph $H'$.
Then, **for each** hyperedge $e$ of $H$:

- **if** $\forall C \subseteq e$: $C$ is not the core of a $(k+1)$-petal sunflower in $H'$
  - **add** $e$ to $H'$.

Unfortunately NP-hard to decide.

# Growing sunflowers

Given a $d$-HS instance $(H, k)$, create an empty hypergraph $H'$.
Then, **for each** hyperedge $e$ of $H$:

- **if** $\forall C \subseteq e\colon |\text{sunflower}[C]| \leq k$, **then**
  - **add** $e$ to $H'$.

# Growing sunflowers

Given a $d$-HS instance $(H, k)$, create an empty hypergraph $H'$.
Then, **for each** hyperedge $e$ of $H$:

- **if** $\forall C \subseteq e\colon |\text{sunflower}[C]| \leq k$, **then**
  - **add** $e$ to $H'$.
  - **for each** $C \subseteq e$, **if** $(e \cup \text{sunflower}[C])$ is a sunflower, **then**
    - add $e$ to sunflower$[C]$

# Growing sunflowers

Given a $d$-HS instance $(H, k)$, create an empty hypergraph $H'$.
Then, **for each** hyperedge $e$ of $H$:

- **if** $\forall C \subseteq e\colon |\text{sunflower}[C]| \leq k$, **then**
  - **add** $e$ to $H'$.
  - **for each** $C \subseteq e$, **if** $(e \cup \text{sunflower}[C])$ is a sunflower, **then**
    - add $e$ to sunflower$[C]$
    - **for each** $v \in e \setminus C$, **set** used$[C][v] \leftarrow$ true

# Growing sunflowers

Given a $d$-HS instance $(H, k)$, create an empty hypergraph $H'$.
Then, **for each** hyperedge $e$ of $H$:

- **if** $\forall C \subseteq e\colon |\text{sunflower}[C]| \leq k$, **then**
  - **add** $e$ to $H'$.
  - **for each** $C \subseteq e$, **if** $\forall v \in e \setminus C\colon \text{used}[C][v] = \text{false}$, **then**
    - add $e$ to sunflower$[C]$
    - **for each** $v \in e \setminus C$, **set** $\text{used}[C][v] \leftarrow \text{true}$

# Growing sunflowers

Given a $d$-HS instance $(H, k)$, create an empty hypergraph $H'$.
Then, **for each** hyperedge $e$ of $H$:

- **if** $\forall C \subseteq e\colon |\text{sunflower}[C]| \leq k$, **then**
  - **add** $e$ to $H'$.
  - **for each** $C \subseteq e$, **if** $\forall v \in e \setminus C\colon \text{used}[C][v] = \text{false}$, **then**
    - add $e$ to sunflower$[C]$
    - **for each** $v \in e \setminus C$, **set** $\text{used}[C][v] \leftarrow \text{true}$

**sunflower[*C*]** and **used[*C*]** can be accessed in $O(d) \subseteq O(1)$ time using a **prefix tree**.

# Growing sunflowers

Given a $d$-HS instance $(H, k)$, create an empty hypergraph $H'$.
Then, **for each** hyperedge $e$ of $H$:

- **if** $\forall C \subseteq e\colon |\text{sunflower}[C]| \leq k$, **then**
  - **add** $e$ to $H'$.
  - **for each** $C \subseteq e$, **if** $\forall v \in e \setminus C\colon \text{used}[C][v] = \text{false}$, **then**
    - add $e$ to sunflower$[C]$
    - **for each** $v \in e \setminus C$, **set** $\text{used}[C][v] \leftarrow \text{true}$

**sunflower[*C*]** and **used[*C*]** can be accessed in $O(d) \subseteq O(1)$ time using a **prefix tree**.

**Running time:** $O(d|V| + 2^d d|E|)$.
**Memory:** $\Theta(|V| \cdot |E|)$.

# Growing sunflowers

Given a $d$-HS instance $(H, k)$, create an empty hypergraph $H'$.
Then, **for each** hyperedge $e$ of $H$:

- **if** $\forall C \subseteq e\colon |\text{sunflower}[C]| \leq k$, **then**
  - **add** $e$ to $H'$.
  - **for each** $C \subseteq e$, **if** $\forall v \in e \setminus C\colon \text{used}[C][v] = \text{false}$, **then**
    - add $e$ to sunflower$[C]$
    - **for each** $v \in e \setminus C$, **set** used$[C][v] \leftarrow$ true

**sunflower[*C*]** and **used[*C*]** can be accessed in $O(d) \subseteq O(1)$ time using a **prefix tree**.

**Running time:** $O(d|V| + 2^d d|E|)$.
**Memory:** $\Theta(|V| \cdot |E|)$.

**Prefix tree** has to be initialized **incompletely** and **carefully**.

# Further results on $d$-Hitting Set

## Experimental results

- Instances from a problem arising in radio frequency allocation.

  [Sorge, Moser, Niedermeier, and Weller, Conference on Integer Programming and Combinatorial Optimization 2012]

  [Babcock, Bell Systems Technical Journal 32, 1953]

# Further results on $d$-Hitting Set

## Experimental results

- Instances from a problem arising in radio frequency allocation.
  [Sorge, Moser, Niedermeier, and Weller, Conference on Integer Programming and Combinatorial Optimization 2012]
  [Babcock, Bell Systems Technical Journal 32, 1953]
- 4-HS with $20 \cdot 10^6$ hyperedges processed in about five minutes.

# Further results on $d$-Hitting Set

## Experimental results

- Instances from a problem arising in radio frequency allocation.
  [Sorge, Moser, Niedermeier, and Weller, Conference on Integer Programming and Combinatorial Optimization 2012]

  [Babcock, Bell Systems Technical Journal 32, 1953]
- 4-HS with $20 \cdot 10^6$ hyperedges processed in about five minutes.
- Prefix trees for **sunflower[]** and **used[]** are faster, but balanced trees have **linear memory usage**.

# Further results on $d$-Hitting Set

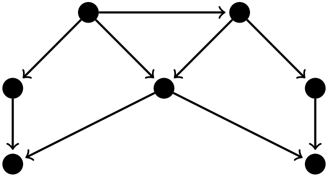
## Experimental results

- Instances from a problem arising in radio frequency allocation.
  [Sorge, Moser, Niedermeier, and Weller, Conference on Integer Programming and Combinatorial Optimization 2012]
  [Babcock, Bell Systems Technical Journal 32, 1953]
- 4-HS with $20 \cdot 10^6$ hyperedges processed in about five minutes.
- Prefix trees for **sunflower[]** and **used[]** are faster, but balanced trees have **linear memory usage**.

**Speed up** kernels of Abu-Khzam and Moser:

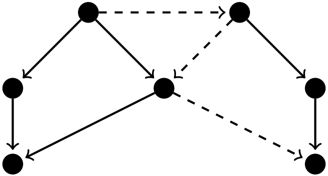
- $O(k^{d-1})$-vertex problem kernel in $O(|V| + |E| + k^{1.5d})$ time.
  [Abu-Khzam, Journal of Computer and System Sciences 76, 2010]
  [Moser, Dissertation, Friedrich-Schiller-Universität Jena, 2010]

# Dag Partitioning



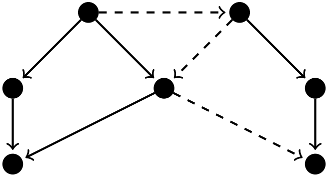
**Task:** Remove at **most $k = 3$ arcs** from a **directed acyclic graph (dag)** so that every vertex **reaches** exactly one sink.

# Dag Partitioning



**Task:** Remove at **most $k = 3$ arcs** from a **directed acyclic graph (dag)** so that every vertex **reaches** exactly one sink.

# Dag Partitioning

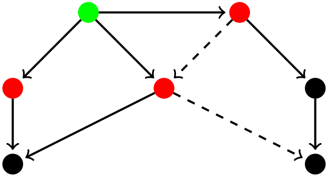


**Task:** Remove at **most $k = 3$ arcs** from a **directed acyclic graph (dag)** so that every vertex **reaches** exactly one sink.

**Algorithmic results:**

- $O(2^k \cdot (n + m))$ time algorithm.
- Linear-time data reduction.

# Dag Partitioning



**Task:** Remove at **most $k = 3$ arcs** from a **directed acyclic graph (dag)** so that every vertex **reaches** exactly one sink.

**Algorithmic results:**

- $O(2^k \cdot (n + m))$ time algorithm.
- Linear-time data reduction.

**Strategy:** First solve problem for all out-neighbors of $v$, then for $v$.

# Experimental results for Dag Partitioning

$O(2^k \cdot (n + m))$ time algorithm on simulated citation networks:

# Experimental results for Dag Partitioning

$O(2^k \cdot (n + m))$ time algorithm on simulated citation networks:

- Instances with $m \geq 10^7$ arcs and $k \leq 190$ arc deletions solvable in five minutes.

# Experimental results for Dag Partitioning

$O(2^k \cdot (n + m))$ time algorithm on simulated citation networks:

- Instances with $m \geq 10^7$ arcs and $k \leq 190$ arc deletions solvable in five minutes.
- Without the linear-time data reduction, no instance solvable in less than an hour.

# Experimental results for Dag Partitioning

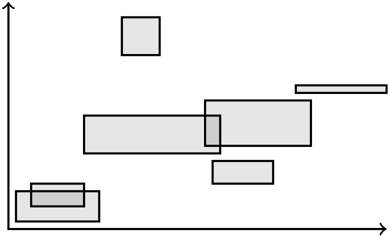
$O(2^k \cdot (n+m))$ time algorithm on simulated citation networks:

- Instances with $m \geq 10^7$ arcs and $k \leq 190$ arc deletions solvable in five minutes.
- Without the linear-time data reduction, no instance solvable in less than an hour.

**Comparison with known heuristic.**
[Leskovec, Backstrom, and Kleinberg, ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2009]

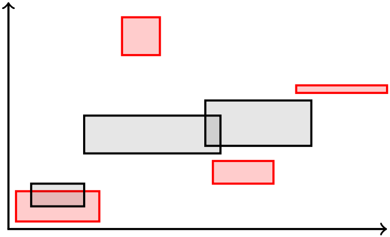
- Heuristic more than a factor of 2.5 off the optimum.
- Algorithm only runs fast where the heuristic gives optimal solutions.

# 2-Union Independent Set



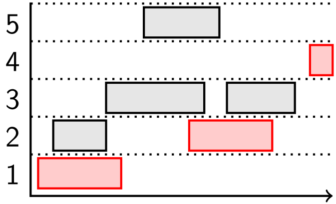
**Task:** Find **at least $k = 4$** rectangles whose projections onto neither axis intersect.

# 2-Union Independent Set



**Task:** Find **at least $k = 4$** rectangles whose projections onto neither axis intersect.

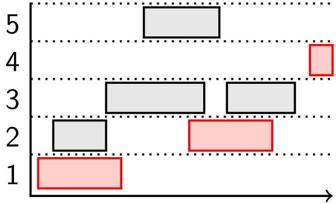
# Job Interval Selection




**Task:** Find **at least $k = 3$** rectangles not intersecting on any axis.

**Special case:** Each rectangle vertically fills **one** of $\gamma$ **strips**.

# Job Interval Selection



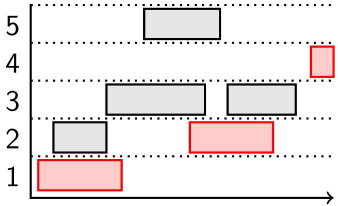

**Task:** Find **at least $k = 3$** rectangles not intersecting on any axis.

**Special case:** Each rectangle vertically fills **one** of $\gamma$ **strips**.

**Known:** Solvable in $O(2^\gamma \gamma \cdot n)$ time.

[Halldórsson and Karlsson, International Workshop on Graph-Theoretic Concepts in Computer Science 2006]

# Job Interval Selection



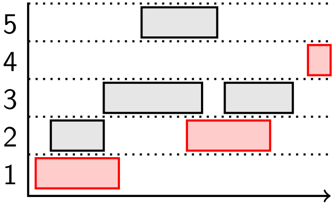

**Task:** Find **at least $k = 3$** rectangles not intersecting on any axis.

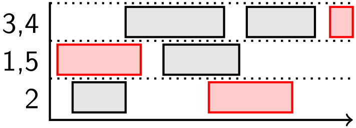
**Special case:** Each rectangle vertically fills **one** of $\gamma$ **strips**.

**Known:** Solvable in $O(2^\gamma \gamma \cdot n)$ time.
[Halldórsson and Karlsson, International Workshop on Graph-Theoretic Concepts in Computer Science 2006]

**New:** $O(5.5^k k \cdot n)$ time, where $k \leq \gamma$.

# Job Interval Selection




**Task:** Find **at least $k = 3$** rectangles not intersecting on any axis.

**Special case:** Each rectangle vertically fills **one** of **$\gamma$ strips**.

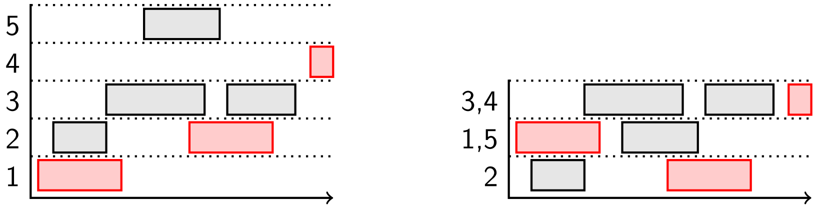
**Known:** Solvable in $O(2^\gamma \gamma \cdot n)$ time.
[Halldórsson and Karlsson, International Workshop on Graph-Theoretic Concepts in Computer Science 2006]

**New:** $O(5.5^k k \cdot n)$ time, where $k \leq \gamma$.

**Trick:** Randomly move strip $i \in \{1, \ldots, \gamma\}$ to strip $j \in \{1, \ldots, k\}$.

- Then use $O(2^\gamma \gamma \cdot n)$-time algorithm with $\gamma = k$.

# Job Interval Selection




**Task:** Find **at least $k = 3$** rectangles not intersecting on any axis.

**Special case:** Each rectangle vertically fills **one** of **$\gamma$ strips**.

**Known:** Solvable in $O(2^\gamma \gamma \cdot n)$ time.

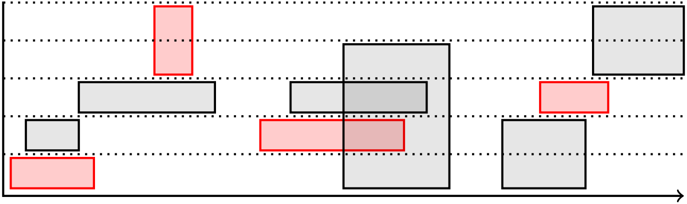
[Halldórsson and Karlsson, International Workshop on Graph-Theoretic Concepts in Computer Science 2006]

**New:** $O(5.5^k k \cdot n)$ time, where $k \leq \gamma$.

**Trick:** Randomly move strip $i \in \{1, \ldots, \gamma\}$ to strip $j \in \{1, \ldots, k\}$.

- Then use $O(2^\gamma \gamma \cdot n)$-time algorithm with $\gamma = k$.
- Repeat $O(e^k \cdot |\ln \varepsilon|)$ times for error probability $\leq \varepsilon$.

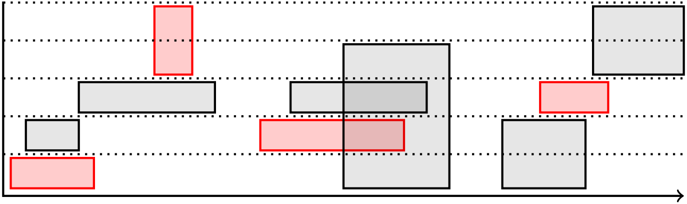
# Compact 2-Union Independent Set



**Task:** Find **at least $k$** rectangles not intersecting on any axis.

**Special case:** Each rectangle vertically fills **any subset** of $\gamma$ **strips** (we say that one axis is $\gamma$**-compact**).

# Compact 2-Union Independent Set



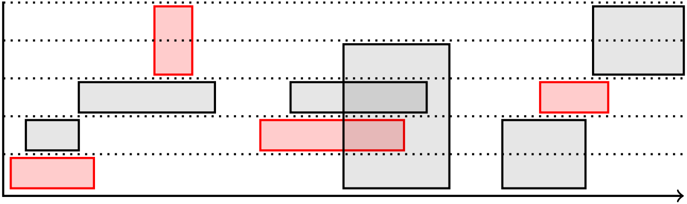
**Task:** Find **at least $k$** rectangles not intersecting on any axis.

**Special case:** Each rectangle vertically fills **any subset** of $\gamma$ **strips** (we say that one axis is **$\gamma$-compact**).

**New:** $O(2^\gamma \gamma \cdot n)$ time algorithm.

- Generalization of Halldórsson and Karlsson's algorithm.

# Compact 2-Union Independent Set



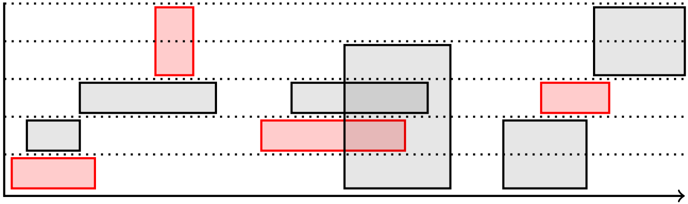
**Task:** Find **at least $k$** rectangles not intersecting on any axis.

**Special case:** Each rectangle vertically fills **any subset** of $\gamma$ **strips** (we say that one axis is $\gamma$**-compact**).

**New:** $O(2^\gamma\gamma \cdot n)$ time algorithm.

- Generalization of Halldórsson and Karlsson's algorithm.
- $\gamma$-compact representation for minimum $\gamma$ in linear time.

# Compact 2-Union Independent Set



**Task:** Find **at least $k$** rectangles not intersecting on any axis.

**Special case:** Each rectangle vertically fills **any subset** of $\gamma$ **strips** (we say that one axis is **$\gamma$-compact**).

**New:** $O(2^\gamma \gamma \cdot n)$ time algorithm.

- Generalization of Halldórsson and Karlsson's algorithm.
- $\gamma$-compact representation for minimum $\gamma$ in linear time.

**Experiments** on random rectangles:

- $n \leq 0.6 \cdot 10^6$ rectangles in $\gamma \leq 15$ strips solved in five minutes.

# Conclusion

**Aspects** brought into fixed-parameter algorithms:

- Returning to the initial goal—efficient algorithms.
- Data structures are suddenly important.

# Conclusion

**Aspects** brought into fixed-parameter algorithms:

- Returning to the initial goal—efficient algorithms.
- Data structures are suddenly important.

**Insights from experiments:**

- Simpler algorithms.
- Provable speed-ups.
- We are on the right way.

# Conclusion

**Aspects** brought into fixed-parameter algorithms:

- Returning to the initial goal—efficient algorithms.
- Data structures are suddenly important.

**Insights from experiments:**

- Simpler algorithms.
- Provable speed-ups.
- We are on the right way.

**Challenges:**

- Parameter race for fixed-parameter linear-time algorithms.

# Conclusion

**Aspects** brought into fixed-parameter algorithms:

- Returning to the initial goal—efficient algorithms.
- Data structures are suddenly important.

**Insights from experiments:**

- Simpler algorithms.
- Provable speed-ups.
- We are on the right way.

**Challenges:**

- Parameter race for fixed-parameter linear-time algorithms.
- Linear-time transformations to problems with well-tuned solvers (SAT, ILP)

# Conclusion

**Aspects** brought into fixed-parameter algorithms:

- Returning to the initial goal—efficient algorithms.
- Data structures are suddenly important.

**Insights from experiments:**

- Simpler algorithms.
- Provable speed-ups.
- We are on the right way.

**Challenges:**

- Parameter race for fixed-parameter linear-time algorithms.
- Linear-time transformations to problems with well-tuned solvers (SAT, ILP) + linear-time data reduction.

# Conclusion

**Aspects** brought into fixed-parameter algorithms:

- Returning to the initial goal—efficient algorithms.
- Data structures are suddenly important.

**Insights from experiments:**

- Simpler algorithms.
- Provable speed-ups.
- We are on the right way.

**Challenges:**

- Parameter race for fixed-parameter linear-time algorithms.
- Linear-time transformations to problems with well-tuned solvers (SAT, ILP) + linear-time data reduction.
- Lower bounds.