

Towards Optimal and Expressive Kernelization for d -Hitting Set

René van Bevern

Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany

63. Theorietag
FH Brandenburg

***d*-Hitting Set and Applications**

***d*-Hitting Set (*d*-HS)**

Input A family E of sets of cardinality at most d over a universe V and a natural number k .

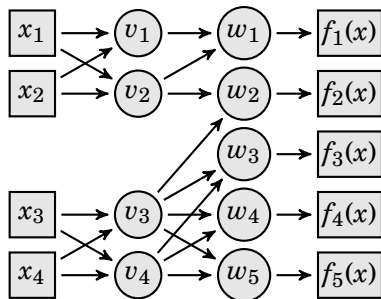
Question Is there a *hitting set* $S \subseteq V$ with $|S| \leq k$ and $\forall e \in E: e \cap S \neq \emptyset$?

The pair $H = (V, E)$ is a *hypergraph*

NP-hard problems solvable by *d*-HS Set include

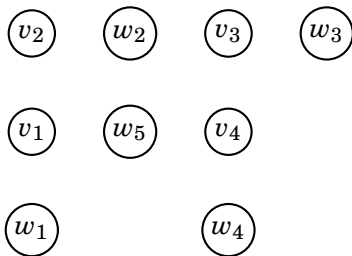
- ▶ graph-based data clustering, e. g. Cluster Vertex Deletion
- ▶ fault diagnosis (of multiple points of failure)

Example: Fault Diagnosis



Boolean circuit: circles are gates, squares are input / output nodes

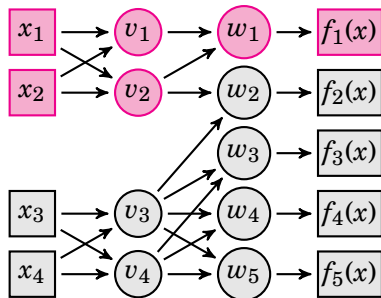
Assume: all outputs wrong



Corresponding hypergraph: an edge contains at least one broken gate

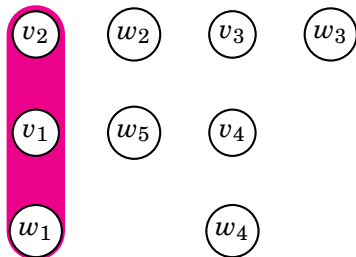
Find minimum hitting set

Example: Fault Diagnosis



Boolean circuit: circles are gates, squares are input / output nodes

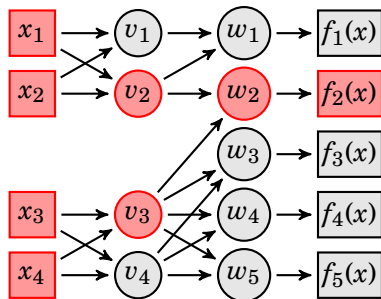
Assume: all outputs wrong



Corresponding hypergraph: an edge contains at least one broken gate

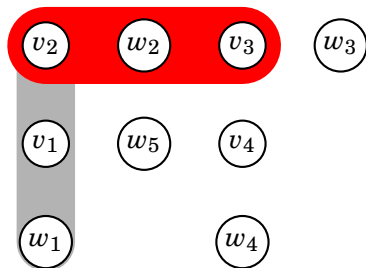
Find minimum hitting set

Example: Fault Diagnosis



Boolean circuit: circles are gates, squares are input / output nodes

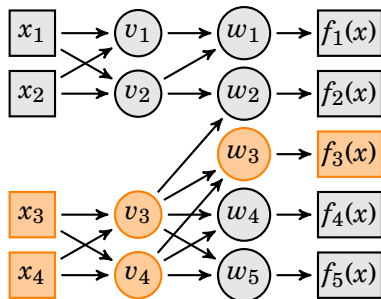
Assume: all outputs wrong



Corresponding hypergraph: an edge contains at least one broken gate

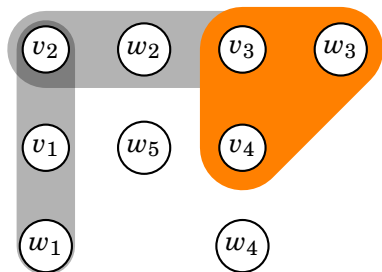
Find minimum hitting set

Example: Fault Diagnosis



Boolean circuit: circles are gates, squares are input / output nodes

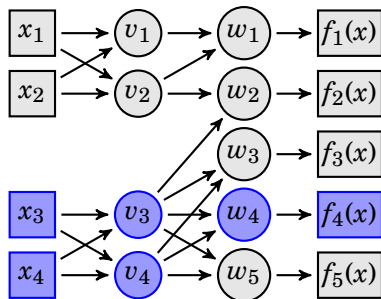
Assume: all outputs wrong



Corresponding hypergraph: an edge contains at least one broken gate

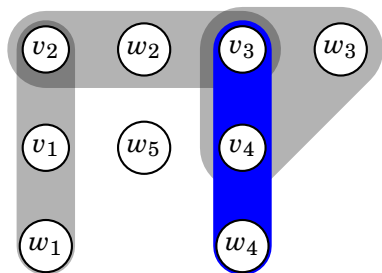
Find minimum hitting set

Example: Fault Diagnosis



Boolean circuit: circles are gates, squares are input / output nodes

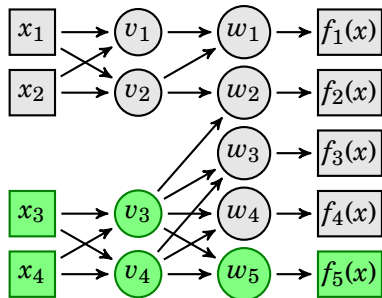
Assume: all outputs wrong



Corresponding hypergraph: an edge contains at least one broken gate

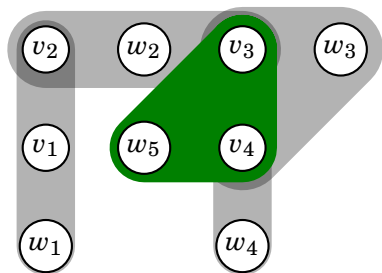
Find minimum hitting set

Example: Fault Diagnosis



Boolean circuit: circles are gates, squares are input / output nodes

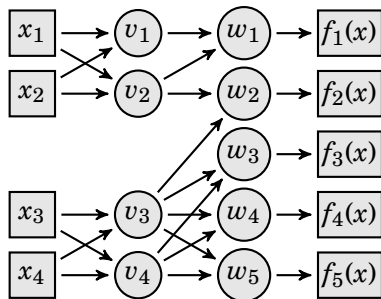
Assume: all outputs wrong



Corresponding hypergraph: an edge contains at least one broken gate

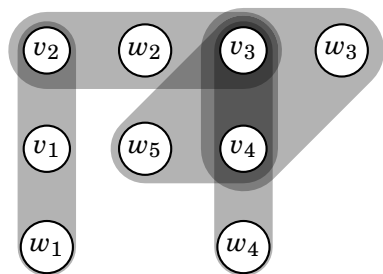
Find minimum hitting set

Example: Fault Diagnosis



Boolean circuit: circles are gates, squares are input / output nodes

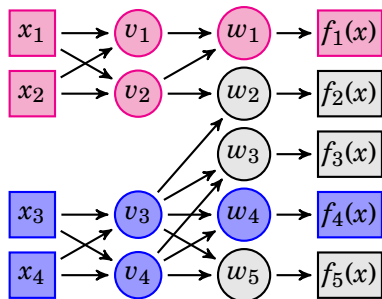
Assume: all outputs wrong



Corresponding hypergraph: an edge contains at least one broken gate

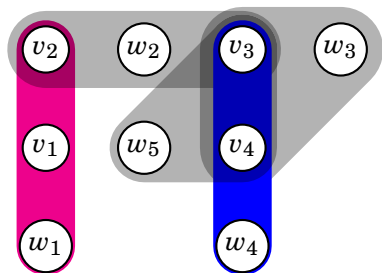
Find minimum hitting set

Example: Fault Diagnosis



Boolean circuit: circles are gates, squares are input / output nodes

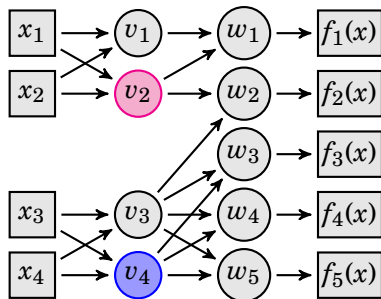
Assume: all outputs wrong



Corresponding hypergraph: an edge contains at least one broken gate

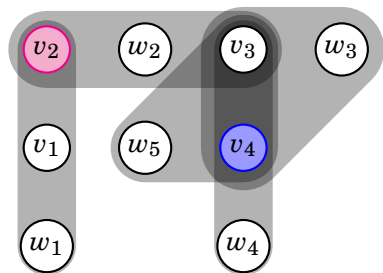
Find minimum hitting set

Example: Fault Diagnosis



Boolean circuit: circles are gates, squares are input / output nodes

Assume: all outputs wrong

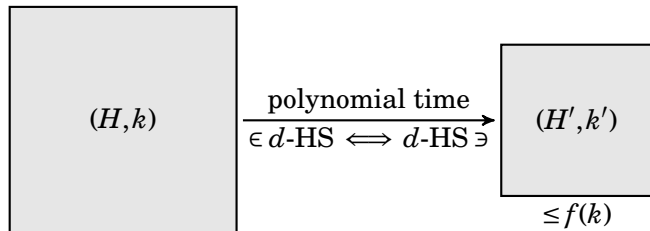


Corresponding hypergraph: an edge contains at least one broken gate

Find minimum hitting set

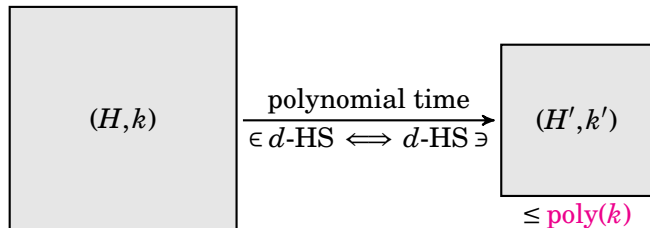
Problem Kernelization

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



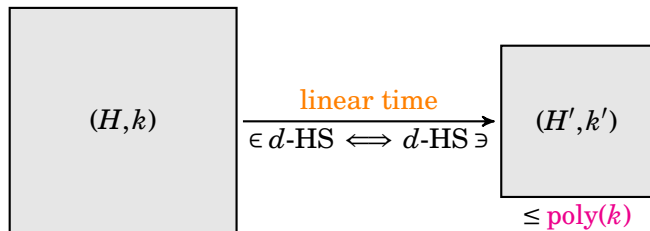
Problem Kernelization

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



Problem Kernelization

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



Known d -HS Kernelization Results

$O(k^{d-\epsilon})$ -size problem kernel \rightsquigarrow polynomial hierarchy collapses

[Dell and van Melkebeek, STOC 2010]

$O(k^3)$ -size problem kernel for 3-HS in **linear time**

[Niedermeier and Rossmanith, J. Discrete Algorithms, 2003]

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

[Flum and Grohe, Book, 2006], [Kratsch, STACS 2009]

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

[Abu-Khzam, WADS 2007], [Moser, Dissertation, 2010]

Known d -HS Kernelization Results

$O(k^{d-\epsilon})$ -size problem kernel \rightsquigarrow polynomial hierarchy collapses

[Dell and van Melkebeek, STOC 2010]

$O(k^3)$ -size problem kernel for 3-HS in linear time

[Niedermeier and Rossmanith, J. Discrete Algorithms, 2003]

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

[Flum and Grohe, Book, 2006], [Kratsch, STACS 2009]

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

[Abu-Khzam, WADS 2007], [Moser, Dissertation, 2010]

Known d -HS Kernelization Results

$O(k^{d-\epsilon})$ -size problem kernel \rightsquigarrow polynomial hierarchy collapses

[Dell and van Melkebeek, STOC 2010]

$O(k^3)$ -size problem kernel for 3-HS in linear time

[Niedermeier and Rossmanith, J. Discrete Algorithms, 2003]

$O(k^d)$ -size expressive problem kernel for d -HS in poly-time

[Flum and Grohe, Book, 2006], [Kratsch, STACS 2009]

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

[Abu-Khzam, WADS 2007], [Moser, Dissertation, 2010]

Improvements

Known Results

$O(k^d)$ -size **expressive** problem kernel for d -HS in poly-time

[Flum Grohe, Book, 2006], [Kratsch, STACS 2009]

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

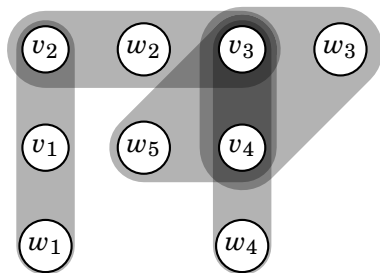
[Abu-Khzam, J. Comput. Syst. Sci., 2010]

Our Results

$O(k^d)$ -size **expressive** problem kernel for d -HS in **linear time**

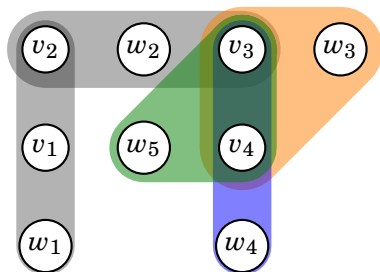
$O(k^{d-1})$ -vertex problem kernel for d -HS in **$O(|V| + |E| + k^{1.5d})$ time**

What do we mean by “expressive”?



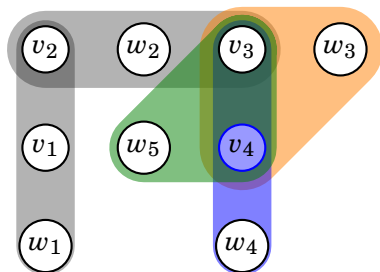
We analyze our fault diagnosis problem.

What do we mean by “expressive”?



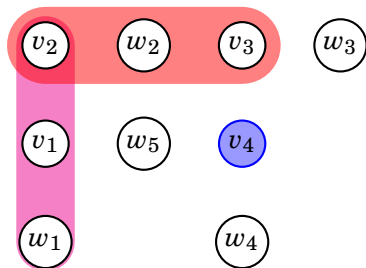
Either one of v_3 or v_4 is broken, or at least three other gates

What do we mean by “expressive”?



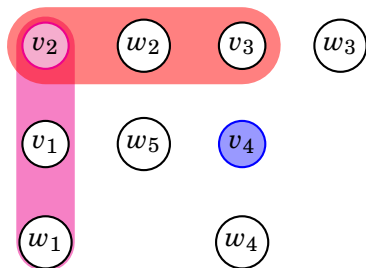
Measurement of the gates can show us that v_4 is broken

What do we mean by “expressive”?



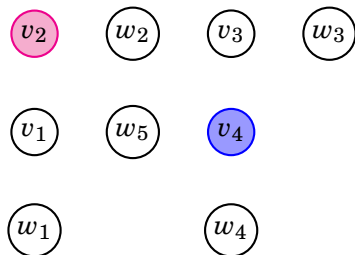
Two broken outputs remain, we search for another broken gate

What do we mean by “expressive”?



Gate v_2 is broken

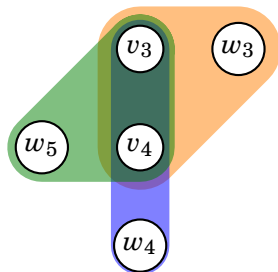
What do we mean by “expressive”?



Gate v_2 is broken

Sunflowers

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



Sunflowers are **expressive**: a compact certificate to show us

- ▶ which gates to check,
- ▶ alternative solutions

Sunflowers II

If no large sunflower can be found, then the input is small enough to be examined by hand or an exponential-time algorithm:

Theorem. *Let $k, d \in \mathbb{N}$ and let H be a d -uniform hypergraph with more than $(k - 1)^d \cdot d!$ hyperedges. Then a sunflower of cardinality k in H can be found in **polynomial time**.*

[Erdős and Rado 1960, Flum and Grohe 2006]

Expressive Kernelization Using Sunflowers

Method 1. Given a d -HS instance (H, k) , **repeatedly**

- ▶ try to **find** a sunflower with $k + 1$ petals,
- ▶ **add** its core to H ,
- ▶ and **remove** its petals from H .

[Flum and Grohe 2006]

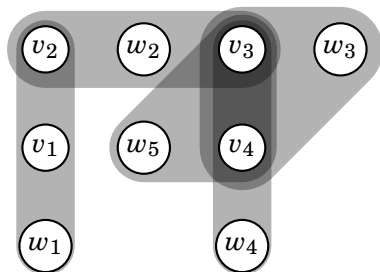
Method 2. Given a d -HS instance (H, k) , **repeatedly**

- ▶ try to **find** a sunflower with $k + 2$ petals,
- ▶ **remove** one of its petals from H .

[Kratsch 2009]

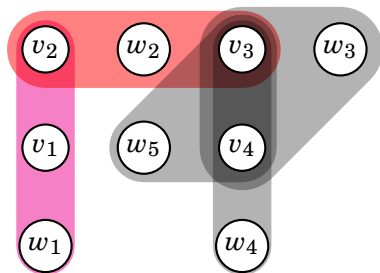
Found sunflowers may guide problem analysis, but data reduction is slow.

Illustration of Kernelization Using Sunflowers



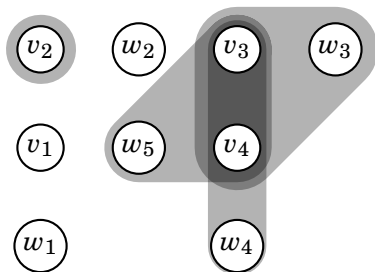
We try to kernelize this instance for $k = 1$

Illustration of Kernelization Using Sunflowers



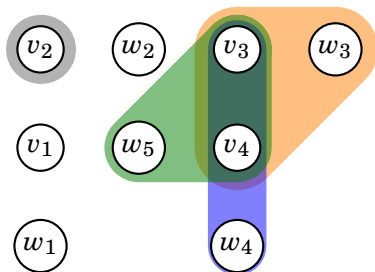
Find sunflower of size $k + 1 = 2$

Illustration of Kernelization Using Sunflowers



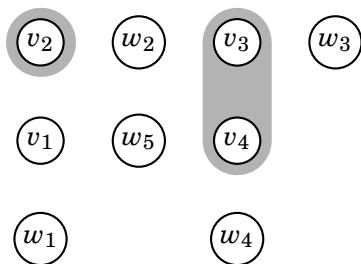
Replace its petals by its core

Illustration of Kernelization Using Sunflowers



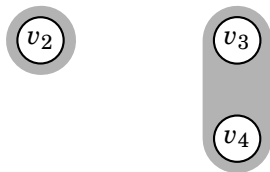
Find sunflower of size $k + 1 = 2$

Illustration of Kernelization Using Sunflowers



Replace its petals by its core

Illustration of Kernelization Using Sunflowers



Resulting problem kernel

Sunflower Kernel in Linear Time

Method 2. Given a d -HS instance (H, k) , **repeatedly**

- ▶ try to **find** a sunflower with $k + 2$ petals,
- ▶ **remove** one of its petals from H .

[Kratsch 2009]

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

- ★ **unless** e contains the core of a $(k + 1)$ -petal sunflower in H'
 - ▶ **add** e to H'

Works in linear time if the test in (★) works in $f(d)$ time.

Sunflower Kernel in Linear Time II

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

- ▶ **unless** e contains the core of a $(k + 1)$ -petal sunflower in H'
 - ▶ **add** e to H'

Sunflower Kernel in Linear Time II

New Method. Given a d -HS instance (H, k) , create an empty hypergraph H' . Then, **for each** edge 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

Sunflower Kernel in Linear Time II

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

- ▶ **if** $\forall C \subseteq e: |\text{petals}[C]| \leq k$, then
 - ▶ **add** e to H'

Sunflower Kernel in Linear Time II

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

- ▶ **if** $\forall C \subseteq e: |\text{petals}[C]| \leq k$, then
 - ▶ **add** e to H'
 - ▶ **for each** $C \subseteq e$, **if** e is valid petal for the sunflower $\text{petals}[C]$,
 - ▶ **add** e to $\text{petals}[C]$

Sunflower Kernel in Linear Time II

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

- ▶ **if** $\forall C \subseteq e: |\text{petals}[C]| \leq k$, then
 - ▶ **add** e to H'
 - ▶ **for each** $C \subseteq e$, **if** e is valid petal for the sunflower $\text{petals}[C]$,
 - ▶ **add** e to $\text{petals}[C]$
 - ▶ **for each** $v \in e \setminus C$, **set** $\text{used}[C][v] \leftarrow \text{true}$

Sunflower Kernel in Linear Time II

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

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

Sunflower Kernel in Linear Time II

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

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

$\text{petals}[C]$ and $\text{used}[C]$ can be accessed in $O(|C|)$ time using a specially linear-time initialized *trie* data structure

Running time: $O(d|V| + 2^d d|E|)$ or $O(d|V| + d|E|^2)$

Memory usage: $O(|V| \cdot |E|)$

Notes

`petals[C]` does not save the maximum sunflower with core C , but a factor- d approximation thereof, hence

- ▶ sunflowers in output graph have at most $d(k + 1)$ petals
- ▶ output graph has at most $d! \cdot d^{d+1} \cdot (k + 1)^d$ edges

[Erdős and Rado 1960, Flum and Grohe 2006]

Similar techniques speed up the kernelization of Abu-Khazam

[Abu-Khazam, J. Comput. Syst. Sci., 2010]

- ▶ $O(k^{d-1})$ -vertex problem kernel in $O(|V| + |E| + k^{1.5d})$ time

The implementation of our algorithm shows:

- ▶ in practice, hash tables for `petals[C]` and `used[C]` save a lot of memory and running time compared to tries

Conclusion

We proved

- ▶ $O(k^d)$ -size problem kernel in $O(|V| + |E|)$ time
- ▶ $O(k^{d-1})$ -vertex problem kernel in $O(|V| + |E| + k^{1.5d})$ time

What about $O(|V| + |E|)$ -time computation of a problem kernel with $O(k^d)$ edges and $O(k^{d-1})$ vertices?

- ▶ problem kernels with $O(k^{d-1})$ vertices involve maximum matching techniques not working in linear time

[Abu-Khzam, J. Comput. Syst. Sci., 2010] [Moser, Dissertation, 2010]

Our running times exponentially depend on d

- ▶ this is also observed in the problem kernel of Abu-Khzam
- ▶ can it be avoided, maintaining linear time?