Clique Is Hard for State-of-the-Art Algorithms

Susanna F. de Rezende

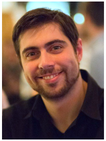
KTH Royal Institute of Technology

Oaxaca
August 2018

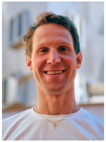
Talk based on joint work with:

A. Atserias

I. Bonacina



M. Lauria

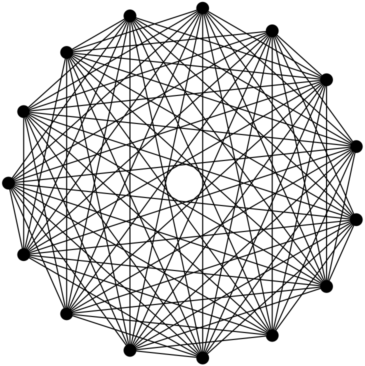


J. Nordström

A. Razborov

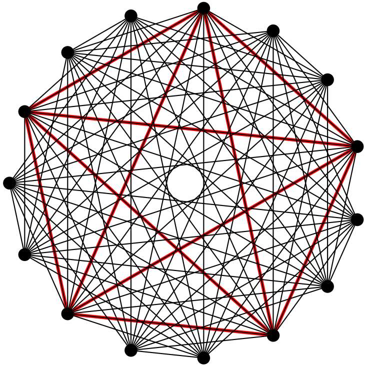
Maximum clique problem

- What is the size of a maximum clique in G?



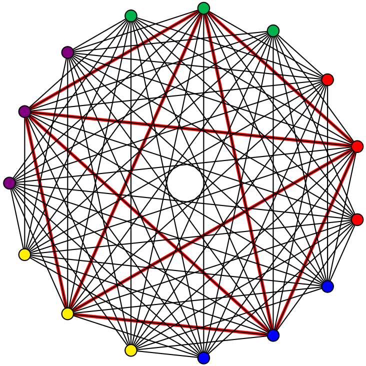
Maximum clique problem

- What is the size of a maximum clique in G?



Maximum clique problem

- What is the size of a maximum clique in G?

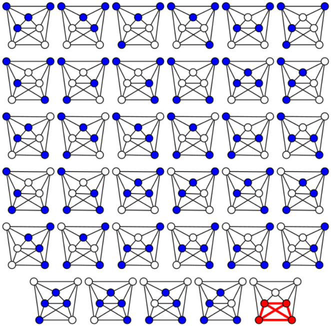


Motivation

- Clique fundamental problem

Motivation

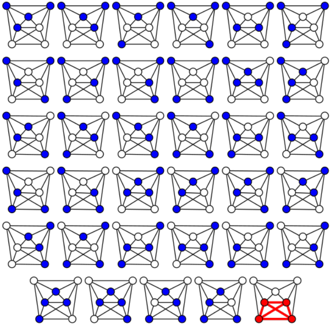
- Clique fundamental problem
- Easy to decide if G contains a k-clique in time n^k



Credit: Thore Husfeldt

Motivation

- Clique fundamental problem
- Easy to decide if G contains a k-clique in time n^k



Credit: Thore Husfeldt

Is this optimal?

Our main result

Theorem (informal)

State-of-the-art algorithms require time $n^{\Omega(k)}$ to determine that the maximum clique in a random graph is k.

Our main result

Theorem (informal)

State-of-the-art algorithms require time $n^{\Omega(k)}$ to determine that the maximum clique in a random graph is k.

- To analyse algorithms need to formalise method of reasoning used

Our main result

Theorem (informal)

State-of-the-art algorithms require time $n^{\Omega(k)}$ to determine that the maximum clique in a random graph is k.

- To analyse algorithms need to formalise method of reasoning used
- If graph has no k-clique, trace of algorithm gives proof of this fact
- Lower bound on size of such proofs $\Rightarrow$ lower bound on running time

Our main result

Theorem (informal)

State-of-the-art algorithms require time $n^{\Omega(k)}$ to determine that the maximum clique in a random graph is k.

- To analyse algorithms need to formalise method of reasoning used
- If graph has no k-clique, trace of algorithm gives proof of this fact
- Lower bound on size of such proofs $\Rightarrow$ lower bound on running time
- Brings us to topic of this talk: proof complexity

What is resolution?

Input: Unsatisfiable CNF formula, e.g.:

$$\neg x \wedge (\neg y \vee \neg z) \wedge (y \vee \neg w) \wedge (x \vee w) \wedge (\neg x \vee z) \wedge \neg y$$

Goal: Certify unsatisfiability using resolution rule

$$\frac{C \vee x \quad D \vee \neg x}{C \vee D}$$

Resolution refutation

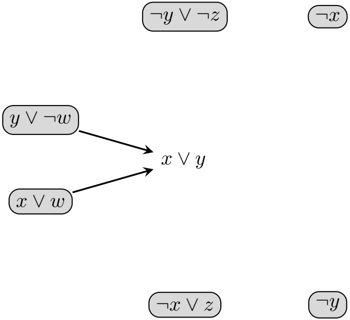
$\neg y \lor \neg z$ $\neg x$

$y \lor \neg w$

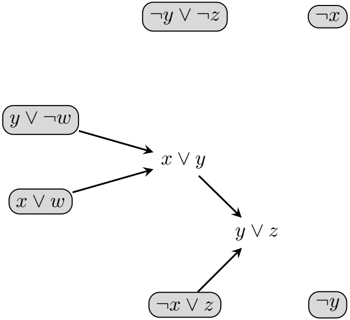
$x \lor w$

$\neg x \lor z$ $\neg y$

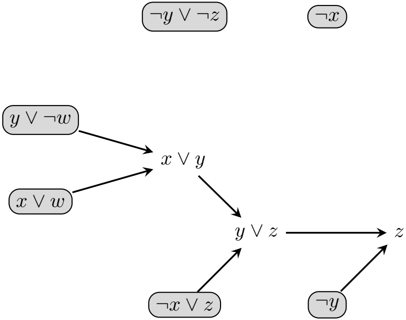
Resolution refutation



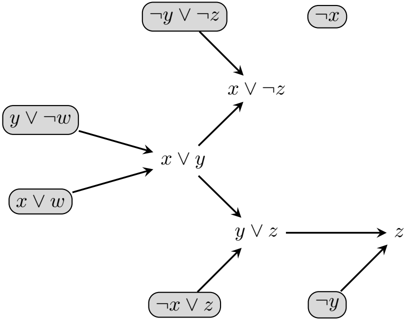
Resolution refutation



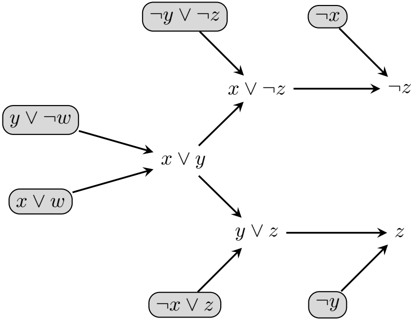
Resolution refutation



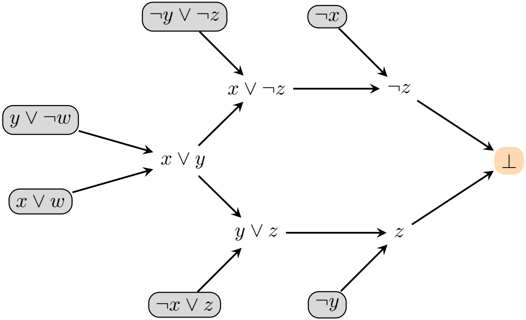
Resolution refutation



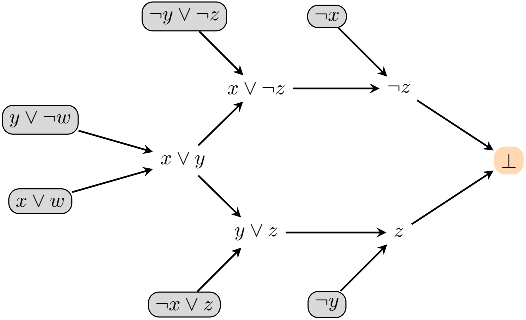
Resolution refutation



Resolution refutation

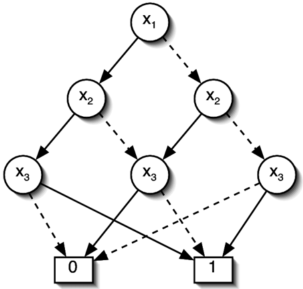


Resolution refutation



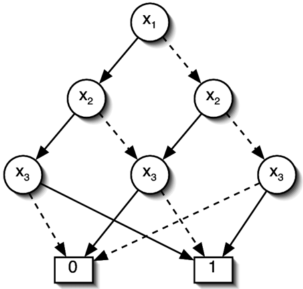
Tree-like = the proof DAG is a tree
Regular = no variable resolved twice in any source-to-sink path
Size = # of nodes in the proof DAG

Branching program



Credit: Airat Khasianov

Branching program

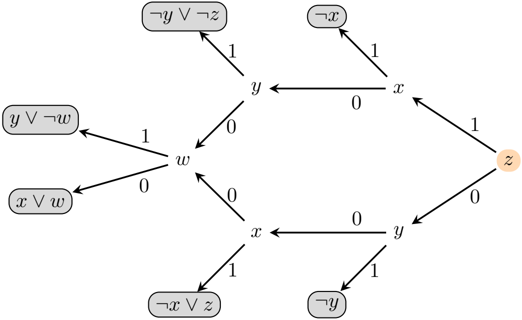


Credit: Airat Khasianov

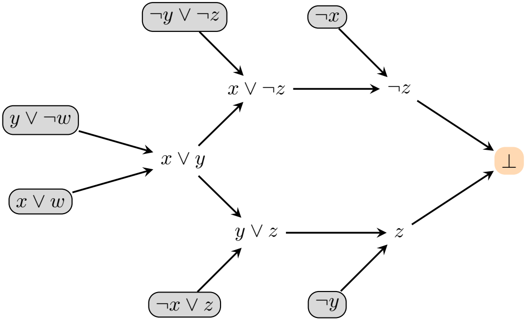
Interested in branching programs solving falsified clause search problem

Falsified clause search problem: given unsat formula and an assignment to variables, find a falsified clause

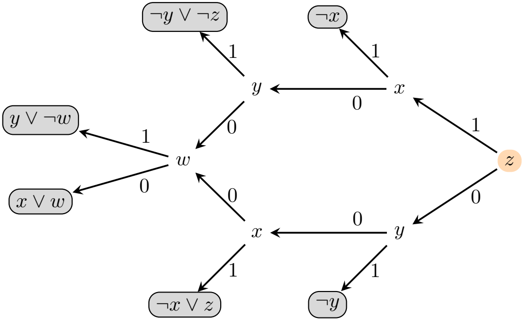
Branching program solving falsified clause problem



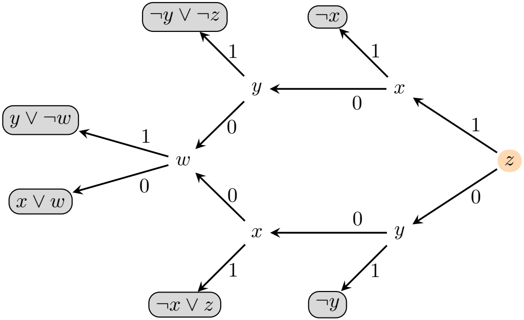
Branching program solving falsified clause problem



Branching program solving falsified clause problem

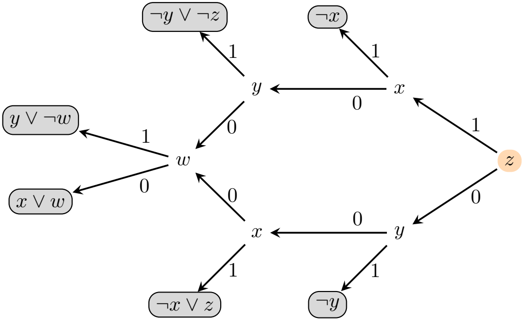


Branching program solving falsified clause problem



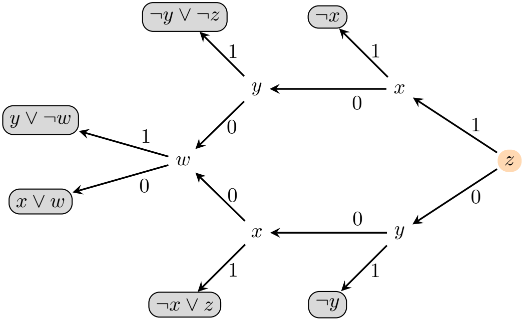
Tree-like = the DAG is a tree
Regular = no variable resolved twice in any source-to-sink path
Size = # of nodes in the DAG

Branching program solving falsified clause problem



Tree-like = the DAG is a tree (a.k.a. decision tree)
Read-once = no variable queried twice in any source-to-sink path
Size = # of nodes in the DAG

Branching program solving falsified clause problem



Decision tree
= tree-like resolution

Read-once branching program = regular resolution

General branching program stronger than general resolution

Tree-like = the DAG is a tree (a.k.a. decision tree)
Read-once = no variable queried twice in any source-to-sink path
Size = # of nodes in the DAG

Encoding the k-clique problem in CNF

Graph $G = (V, E)$, $k \in \mathbb{N}$

Formula $\textbf{Clique}(\boldsymbol{G}, \boldsymbol{k})$:

$x_{v,i}$: "vertex v is i-th member of clique"

Encoding the k-clique problem in CNF

Graph $G = (V, E)$, $k \in \mathbb{N}$

Formula $\mathbf{Clique}(\boldsymbol{G}, \boldsymbol{k})$:

$x_{v,i}$: "vertex v is i-th member of clique"

$\exists i$th clique-member	$\bigvee_{v \in V} x_{v,i}$	$i \in [k]$
non-neighbours are not both in clique	$\neg x_{v,i} \vee \neg x_{u,j}$	$(v, u) \notin E$

Encoding the k-clique problem in CNF

Graph $G = (V, E)$, $k \in \mathbb{N}$

Formula **Clique(G, k)**:

$x_{v,i}$: "vertex v is i-th member of clique"

$\exists i$th clique-member	$\bigvee_{v \in V} x_{v,i}$	$i \in [k]$
non-neighbours are not both in clique	$\neg x_{v,i} \vee \neg x_{u,j}$	$(v, u) \notin E$

Formula Clique(G, k) is satisfiable $\Leftrightarrow$ G has a k-clique

State-of-the-art algorithms

Östergård's algorithm using Russian doll search

- often used in practice
- has been available online since 2003
- main component of the Cliquer software
- algorithm of choice in the open source software SageMath

State-of-the-art algorithms

Östergård's algorithm using Russian doll search

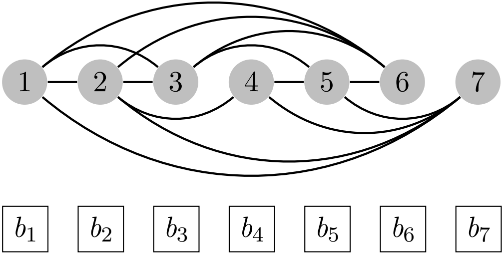
- often used in practice
- has been available online since 2003
- main component of the Cliquer software
- algorithm of choice in the open source software SageMath

Colour-based branch-and-bound algorithms

- arguably the most successful in practice
- uses colouring as bounding (and often as branching) strategy
- basic idea: $(k-1)$-colourable graph cannot contain k-clique
- extended survey and computational analysis in [Prosser'12] and [McCreesh'17]

Östergård's algorithm

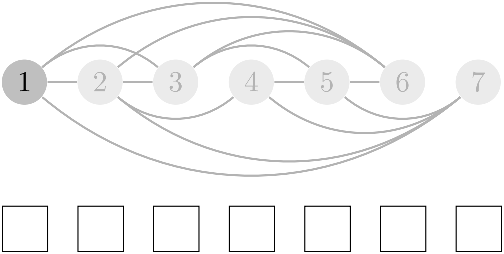
- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$



$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

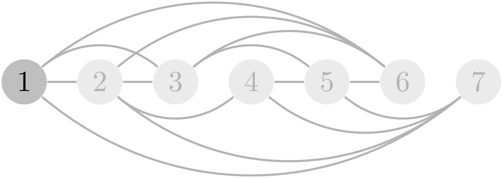
- $G = (V, E)$
- $V = [n] = \{1, 2, \dots, n\}$



$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \dots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

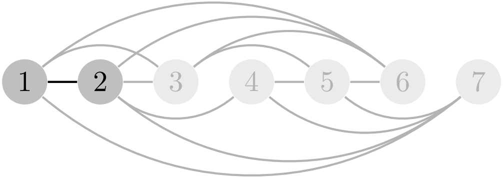


1						

$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

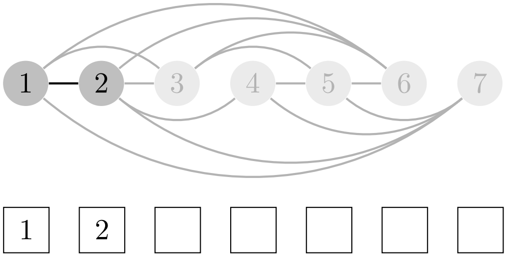


1						

$b_i :=$ $\max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

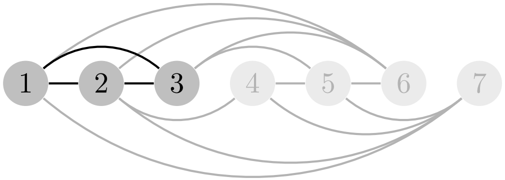
- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$



$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

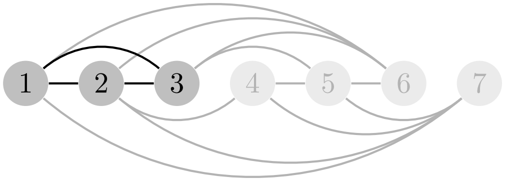


$$\boxed{1} \quad \boxed{2} \quad \boxed{} \quad \boxed{} \quad \boxed{} \quad \boxed{} \quad \boxed{}$$

$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

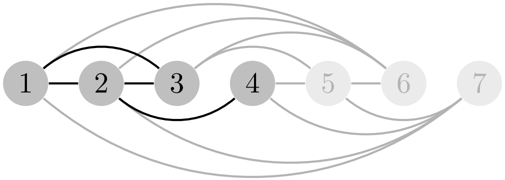


1	2	3				

$b_i :=$ max$\{\ell : \exists\ \ell$-clique among vertices $\{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

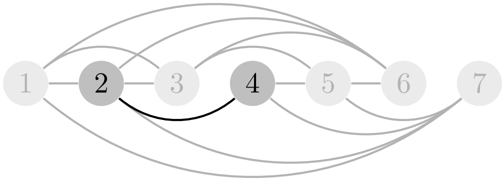


$\boxed{1}$ $\boxed{2}$ $\boxed{3}$ $\boxed{}$ $\boxed{}$ $\boxed{}$ $\boxed{}$

$b_i := \max\{\ell : \exists\ \ell$-clique among vertices $\{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

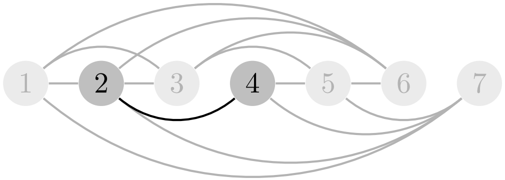


1	2	3				

$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

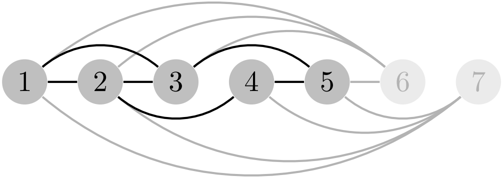


1	2	3	3			

$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

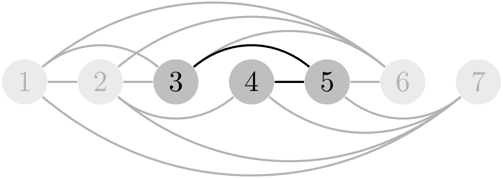


1	2	3	3			

$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

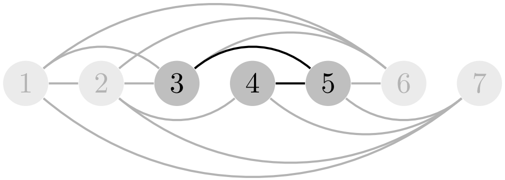


1	2	3	3			

$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

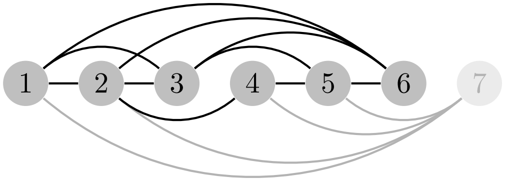


$\boxed{1}$ $\boxed{2}$ $\boxed{3}$ $\boxed{3}$ $\boxed{3}$ $\boxed{}$ $\boxed{}$

$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

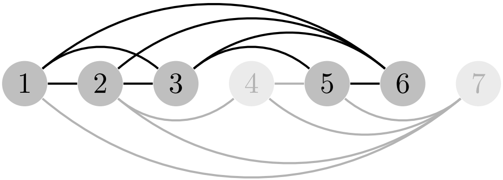


$\boxed{1}$ $\boxed{2}$ $\boxed{3}$ $\boxed{3}$ $\boxed{3}$ $\boxed{}$ $\boxed{}$

$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

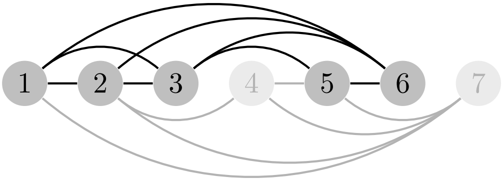


1	2	3	3	3		

$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

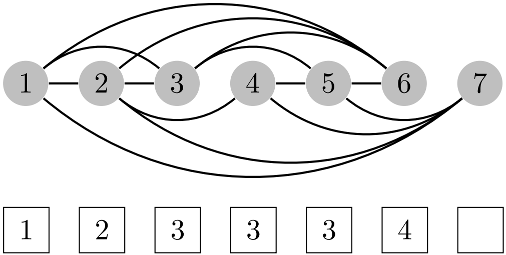


1	2	3	3	3	4	

$b_i :=$ $\max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

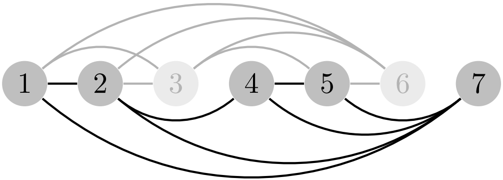
- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$



$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$

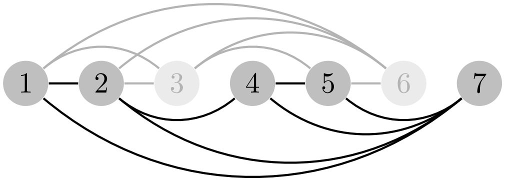


1	2	3	3	3	4	

$b_i :=$ max$\{\ell : \exists\ \ell$-clique among vertices $\{1, 2, \ldots, i\}\}$

Östergård's algorithm

- $G = (V, E)$
- $V = [n] = \{1, 2, \ldots, n\}$



1	2	3	3	3	4	4

$b_i := \max\{\ell : \exists\ \ell\text{-clique among vertices } \{1, 2, \ldots, i\}\}$

Östergård's algorithm

$\texttt{Cliquer}(G)$:
begin
  $G \leftarrow \texttt{permute}(G)$
  $inc \leftarrow \emptyset$
  for $i = n$ ***down to*** 1 **do**
    $found \leftarrow$ false
    $\texttt{expand}(G[V_i \cap N(v_i)], \{v_i\})$
    $b[i] \leftarrow |inc|$
  return inc

$\texttt{expand}(H, sol)$:
begin
  while $V(H) \neq \emptyset$ **do**
    if $|sol| + |V(H)| \leq |inc|$ **then return**
    $i \leftarrow \min\{j \mid v_j \in V(H)\}$
    if $|sol| + b[i] \leq |inc|$ **then return**
    $sol' \leftarrow sol \cup \{v_i\}$
    $V' \leftarrow V(H) \cap N(v_i)$
    $\texttt{expand}(H[V'], sol')$
    if $found =$ true **then return**
    $H \leftarrow H \setminus \{v_i\}$
  if $|sol'| > |inc|$ **then**
    $inc \leftarrow sol'$, $found \leftarrow$ true
  return

Colour-based branch and bound

`MaxClique`(G):
begin
  global $inc \leftarrow \emptyset$
  `expand`$(G, \emptyset)$
  return inc

`expand`(H, sol):
begin
  $(order, b) \leftarrow$ `colourOrder`(H)
  while $V(H) \neq \emptyset$ **do**
    $i \leftarrow |V(H)|$
    if $|sol| + b[i] \leq |inc|$ **then return**
    $v \leftarrow order[i]$
    $sol' \leftarrow sol \cup \{v\}$
    $V' \leftarrow V(H) \cap N(v)$
    `expand`$(H[V'], sol')$
    $H \leftarrow H \setminus \{v\}$
  if $|sol'| > |inc|$ **then** $inc \leftarrow sol'$
  return

What kind of proofs do these algorithms generate?

Recall encoding of k-clique problem in CNF

$x_{v,i}$: "vertex v is i-th member of clique"

$\exists i$th clique-member	$\bigvee_{v \in V} x_{v,i}$	$i \in [k]$
non-neighbours are not both in clique	$\neg x_{v,i} \vee \neg x_{u,j}$	$(v,u) \notin E$

Recall encoding of k-clique problem in CNF

$x_{v,i}$: "vertex v is i-th member of clique"

$\exists i$th clique-member	$\bigvee_{v \in V} x_{v,i}$	$i \in [k]$
non-neighbours are not both in clique	$\neg x_{v,i} \vee \neg x_{u,j}$	$(v,u) \notin E$

Doesn't capture: if no k-clique where v is 1st member $\Rightarrow$
no k-clique where v is 2nd member

Recall encoding of k-clique problem in CNF

$x_{v,i}$: "vertex v is i-th member of clique"

$\exists i$th clique-member	$\bigvee_{v \in V} x_{v,i}$	$i \in [k]$
non-neighbours are not both in clique	$\neg x_{v,i} \vee \neg x_{u,j}$	$(v,u) \notin E$

Doesn't capture: if no k-clique where v is 1st member $\Rightarrow$
no k-clique where v is 2nd member

Fix: define "stronger" formula
$\Rightarrow$ split $V = V_1 \dot{\cup} \ldots \dot{\cup} V_k$ s.t. $v \in V_i$ can only be ith clique member

Recall encoding of k-clique problem in CNF

x_v: "vertex v is i-th member of clique" for $v \in V_i$

$\exists i$th clique-member	$\bigvee_{v \in V_i} x_v$	$i \in [k]$
non-neighbours are not both in clique	$\neg x_v \vee \neg x_u$	$(v, u) \notin E$

Doesn't capture: if no k-clique where v is 1st member $\Rightarrow$
no k-clique where v is 2nd member

Fix: define "stronger" formula
$\Rightarrow$ split $V = V_1 \dot\cup \ldots \dot\cup V_k$ s.t. $v \in V_i$ can only be ith clique member

Recall encoding of k-clique problem in CNF

x_v: "vertex v is i-th member of clique" for $v \in V_i$

$\exists i$th clique-member	$\bigvee_{v \in V_i} x_v$	$i \in [k]$
non-neighbours are not both in clique	$\neg x_v \vee \neg x_u$	$(v,u) \notin E$

Doesn't capture: if no k-clique where v is 1st member $\Rightarrow$
no k-clique where v is 2nd member

Fix: define "stronger" formula
$\Rightarrow$ split $V = V_1 \dot{\cup} \ldots \dot{\cup} V_k$ s.t. $v \in V_i$ can only be ith clique member

G has no k-clique $\Rightarrow$ formula unsat (converse not necessarily true)

What kind of proofs do these algorithms generate?

Colouring-base branch-and-bound algorithm

- Branch = tree-like resolution

What kind of proofs do these algorithms generate?

Colouring-base branch-and-bound algorithm

- Branch = tree-like resolution
- Bound = ?

What kind of proofs do these algorithms generate?

Colouring-base branch-and-bound algorithm

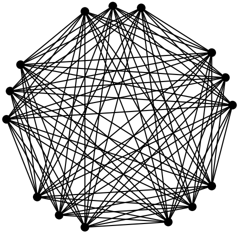
- Branch $=$ tree-like resolution
- Bound $=$?
- Are there small resolution proofs of the fact that $(k-1)$-colourable graphs do not contain k-cliques?

What kind of proofs do these algorithms generate?

Colouring-base branch-and-bound algorithm

- Branch = tree-like resolution
- Bound = ?
- Are there small resolution proofs of the fact that $(k-1)$-colourable graphs do not contain k-cliques?

- Complete $(k-1)$-partite graph

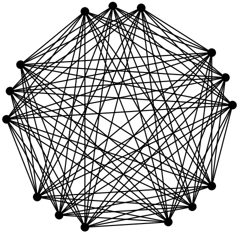


What kind of proofs do these algorithms generate?

Colouring-base branch-and-bound algorithm

- Branch = tree-like resolution
- Bound = ?
- Are there small resolution proofs of the fact that $(k-1)$-colourable graphs do not contain k-cliques?

- Complete $(k-1)$-partite graph
- Hard for tree-like resolution

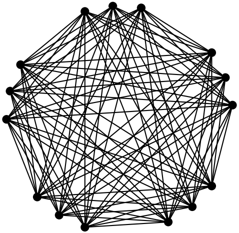


What kind of proofs do these algorithms generate?

Colouring-base branch-and-bound algorithm

- Branch = tree-like resolution
- Bound = ?
- Are there small resolution proofs of the fact that $(k-1)$-colourable graphs do not contain k-cliques?

- Complete $(k-1)$-partite graph
- Hard for tree-like resolution
- **Easy** for regular resolution! (can do $\approx 2^k n^2$)
- In fact, any $(k-1)$-colourable graph is easy for regular resolution

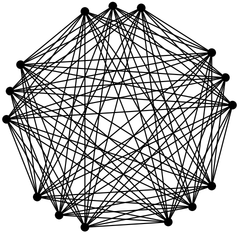


What kind of proofs do these algorithms generate?

Colouring-base branch-and-bound algorithm

- Branch = tree-like resolution
- Bound = regular resolution
- Are there small resolution proofs of the fact that $(k-1)$-colourable graphs do not contain k-cliques?

- Complete $(k-1)$-partite graph
- Hard for tree-like resolution
- **Easy** for regular resolution! (can do $\approx 2^k n^2$)
- In fact, any $(k-1)$-colourable graph is easy for regular resolution



What kind of proofs do these algorithms generate?

Colouring-base branch-and-bound algorithm

- Branch = tree-like resolution
- Bound = regular resolution
- Are there small resolution proofs of the fact that $(k-1)$-colourable graphs do not contain k-cliques?

Östergård's algorithm

- Branch = tree-like resolution
- Bound = regular resolution
- Reuse previous computations for bounding

Regular resolution captures any such algorithms, even for oracle access to optimal ordering of vertices and optimal colourings

Hardness of k-clique for resolution

Previous work

- $n^{\Omega(k)}$ for tree-like resolution [Beyersdorff, Galesi, Lauria '11]
- $n^{\Omega(k)}$ for general resolution for *binary encoding* [Lauria, Pudlák, Rödl, Thapen '13]
- $\exp\left(n^{\Omega(1)}\right)$ for general resolution for $k \gg n^{5/6}$ [Beame, Impagliazzo, Sabharwal '01]

Hardness of k-clique for resolution

Previous work

- $n^{\Omega(k)}$ for tree-like resolution [Beyersdorff, Galesi, Lauria '11]
- $n^{\Omega(k)}$ for general resolution for *binary encoding* [Lauria, Pudlák, Rödl, Thapen '13]
- $\exp\left(n^{\Omega(1)}\right)$ for general resolution for $k \gg n^{5/6}$ [Beame, Impagliazzo, Sabharwal '01]

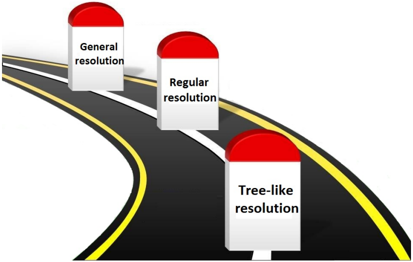
Many reasons to care about small k

Hardness of k-clique for resolution

Previous work

- $n^{\Omega(k)}$ for tree-like resolution [Beyersdorff, Galesi, Lauria '11]
- $n^{\Omega(k)}$ for general resolution for *binary encoding* [Lauria, Pudlák, Rödl, Thapen '13]
- $\exp\left(n^{\Omega(1)}\right)$ for general resolution for $k \gg n^{5/6}$ [Beame, Impagliazzo, Sabharwal '01]

Many reasons to care about small k



Hardness of k-clique for resolution

Previous work

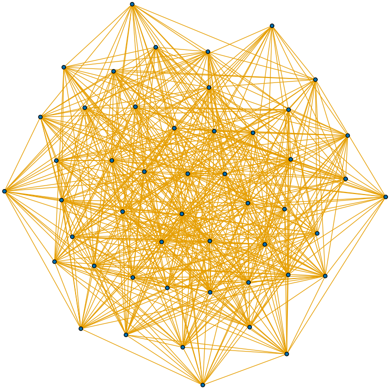
- $n^{\Omega(k)}$ for tree-like resolution [Beyersdorff, Galesi, Lauria '11]
- $n^{\Omega(k)}$ for general resolution for *binary encoding* [Lauria, Pudlák, Rödl, Thapen '13]
- $\exp\left(n^{\Omega(1)}\right)$ for general resolution for $k \gg n^{5/6}$ [Beame, Impagliazzo, Sabharwal '01]

Usual proof complexity tool-box seems to fail:

- Random restrictions
- Interpolation techniques [Krajíček '97]
- Size-width lower bound [Ben-Sasson, Wigderson '99]

What are hard instance for regular resolution?

- Erdős-Rényi random graph: $G \sim \mathcal{G}(n, p)$
- n vertices
- include each possible edge with probability p



What are hard instance for regular resolution?

- Erdős-Rényi random graph: $G \sim \mathcal{G}(n, p)$
- n vertices
- include each possible edge with probability p

What is an appropriate p?

$$E[\#\text{ of } k\text{-cliques}] = \binom{n}{k} p^{k(k-1)/2}$$

- Threshold value for having a k-clique $p \approx n^{-2/(k-1)}$
- Choose p slightly below so that w.h.p. no k-clique but still dense

Slightly more formal statement of main result

Theorem

Let $k \ll n^{1/4}$ and let $G \sim \mathcal{G}(n, p)$ for p slightly less than threshold. W.h.p. any regular resolution refutation of $\mathsf{Clique}(G, k)$ has length $n^{\Omega(k)}$.

Slightly more formal statement of main result

Theorem

Let $k \ll n^{1/4}$ *and let* $G \sim \mathcal{G}(n,p)$ *for* p *slightly less than threshold. W.h.p. any regular resolution refutation of* Clique(G,k) *has length* $n^{\Omega(k)}$.

- Tight: upper bound $n^{O(k)}$ (even for tree-like resolution)
- Lower bound degrades gracefully with smaller density

Take away

Summary

- k-clique fundamental problem
- Prove $n^{\Omega(k)}$ average case lower bound for regular resolution
- Holds for proof system that captures state-of-the-art algorithms

Take away

Summary

- k-clique fundamental problem
- Prove $n^{\Omega(k)}$ average case lower bound for regular resolution
- Holds for proof system that captures state-of-the-art algorithms

Open problems

- Prove hardness for explicit graphs
- Extend to *general* resolution
- Why are CDCL solvers slower than clique-solvers?
- Can we design better algorithms (e.g. that are not captured by regular resolution)?

Take away

Summary

- k-clique fundamental problem
- Prove $n^{\Omega(k)}$ average case lower bound for regular resolution
- Holds for proof system that captures state-of-the-art algorithms

Open problems

- Prove hardness for explicit graphs
- Extend to *general* resolution
- Why are CDCL solvers slower than clique-solvers?
- Can we design better algorithms (e.g. that are not captured by regular resolution)?

Thanks!