A can be computed as in the antimonotone case.

How to compute M?

Let $M(S, T)$ be the number of clauses that have exactly all variables in T as their negative literals and have at least the variables in S as positive literals.

Then

$$M = \sum_{S,T \subseteq P, |S|, |T| \leq s} (-1)^{|S|+1} M(S, T),$$

if P is the set of variables on the tape.

A can be computed as in the antimonotone case.

How to compute M?

Let $M(S, T)$ be the number of clauses that have exactly all variables in T as their negative literals and have at least the variables in S as positive literals.

Then

$$M = \sum_{S,T \subseteq P, |S|,|T| \leq s} (-1)^{|S|+1} M(S, T),$$

if P is the set of variables on the tape.

A can be computed as in the antimonotone case.

How to compute M?

Let $M(S,T)$ be the number of clauses that have exactly all variables in T as their negative literals and have at least the variables in S as positive literals.

Then

$$M = \sum_{S,T \subseteq P, |S|,|T| \leq s} (-1)^{|S|+1} M(S,T),$$

if P is the set of variables on the tape.

Definition

Multicolored Clique is the following problem:
Input: A Graph G, nodes colored by k colors
Parameter: k
Question: Is there a k-clique with k colors?

Theorem

Multicolored Clique is $W[1]$*-hard.*

Definition

List Coloring (parameterized by treewidth) is the following problem:
Input: A Graph G, nodes have lists of colors
Parameter: treewidth of G
Question: Is there a node coloring with colors from the lists?

Theorem

tw–List Coloring is $W[1]$*-hard.*

Definition

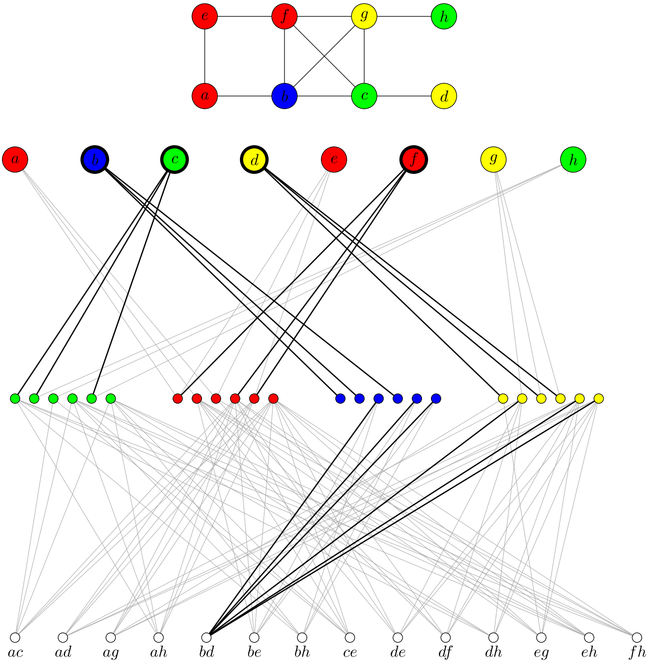
Multicolored Grid is the following problem:
Input: A Graph G, nodes colored by $\{(i,j) \mid 1 \leq i,j \leq k\}$.
Parameter: k
Question: Is there a $k \times k$-grid whose node at coordinates (i,j) is colored with (i,j)?

Theorem

Multicolored Grid is $W[1]$-hard.

Theorem

List coloring is $W[1]$-hard with parameter treewidth even for planar graphs.


e f g h
a b c d
a b c d e f g h
ac ad ag ah bd be bh ce de df dh eg eh fh

Definition

An OR-distillation algorithm for a problem L is an algorithm that transforms $(v_1, \dots, v_t)$ into w with these properties:

1. runs in polynomial time
2. $w \in L$ iff some $v_i \in L$
3. $|w|$ polynomially bounded in $|v_i|$ for all i

For which problems L do distillation algorithms exist?

Theorem

If an OR-distillation algorithm for an NP-complete problem exists, then coNP $\subseteq$ NP/poly.

Definition

An OR-distillation algorithm for a problem L is an algorithm that transforms $(v_1, \ldots, v_t)$ into w with these properties:

1. runs in polynomial time
2. $w \in L$ iff some $v_i \in L$
3. $|w|$ polynomially bounded in $|v_i|$ for all i

For which problems L do distillation algorithms exist?

Theorem

If an OR-distillation algorithm for an NP-complete problem exists, then $coNP \subseteq NP/poly$.

Definition

An OR-composition algorithm for a parameterized problem L is an algorithm that transforms $((v_1, k), \ldots, (v_t, k))$ into (w, k') with these properties:

1. runs in polynomial time
2. $(w, k') \in L$ iff some $(v_i, k) \in L$
3. k' polynomially bounded in k

Theorem

Let L be an NP-complete parameterized problem (where the parameter is encoded in unary as part of the input). If there is an OR-composition algorithm for L and L has a polynomial kernel, then there is also an OR-distillation algorithm for L.

Definition

An OR-composition algorithm for a parameterized problem L is an algorithm that transforms $((v_1, k), \ldots, (v_t, k))$ into (w, k') with these properties:

1. runs in polynomial time
2. $(w, k') \in L$ iff some $(v_i, k) \in L$
3. k' polynomially bounded in k

Theorem

Let L be an NP-complete parameterized problem (where the parameter is encoded in unary as part of the input). If there is an OR-composition algorithm for L and L has a polynomial kernel, then there is also an OR-distillation algorithm for L.

Theorem

If a parameterized problem has an OR-composition algorithm and a polynomial kernel, then coNP $\subseteq$ NP/poly.

The following problems have OR-composition algorithms:

- k-path
- k-cycle

Theorem

If a parameterized problem has an OR-composition algorithm and a polynomial kernel, then coNP $\subseteq$ NP/poly.

The following problems have OR-composition algorithms:

- k-path
- k-cycle

A similar framework exists for AND-composition and AND-distillation.

Theorem

If a parameterized problem has an AND-composition algorithm and a polynomial kernel, then $coNP \subseteq NP/poly$.

The following problems have AND-composition algorithms:

- treewidth
- pathwidth

A similar framework exists for AND-composition and AND-distillation.

Theorem

If a parameterized problem has an AND-composition algorithm and a polynomial kernel, then $coNP \subseteq NP/poly$.

The following problems have AND-composition algorithms:

- treewidth
- pathwidth

Definition

The k-leaf outbranching problem:

- Input: A directed graph G and a number k
- Parameter: k
- Question: Does G have a k-leaf outbranching.

An outbranching is a directed out-tree.

This problem has an OR-composition algorithm, hence no polynomial kernel.

Definition

The k-leaf outbranching problem:

- Input: A directed graph G and a number k
- Parameter: k
- Question: Does G have a k-leaf outbranching.

An outbranching is a directed out-tree.

This problem has an OR-composition algorithm, hence no polynomial kernel.

Definition

The rooted k-leaf outbranching problem:

- Input: A directed graph G, a node r, and a number k
- Parameter: k
- Question: Does G have a k-leaf outbranching with root r.

No easy to find OR-composition algorithm.

Indeed, there is a k^3 kernel for this problem.

(Proof: Very technical with five reduction rules.)

Definition

The rooted k-leaf outbranching problem:

- Input: A directed graph G, a node r, and a number k
- Parameter: k
- Question: Does G have a k-leaf outbranching with root r.

No easy to find OR-composition algorithm.

Indeed, there is a k^3 kernel for this problem.

(Proof: Very technical with five reduction rules.)

Definition

The rooted k-leaf outbranching problem:

- Input: A directed graph G, a node r, and a number k
- Parameter: k
- Question: Does G have a k-leaf outbranching with root r.

No easy to find OR-composition algorithm.

Indeed, there is a k^3 kernel for this problem.

(Proof: Very technical with five reduction rules.)

Can we "reduce" the k-leaf outbranching problem to the rooted k-leaf outbranching problem?

Yes and No. Depends on what "reduce" exactly means.

We can take a k-leaf outbranching problem and reduce it to n instances of rooted k-leaf outbranching.

This is similar to a kernel and is called a "Turing kernel."

Can we "reduce" the k-leaf outbranching problem to the rooted k-leaf outbranching problem?

Yes and No. Depends on what "reduce" exactly means.

We can take a k-leaf outbranching problem and reduce it to n instances of rooted k-leaf outbranching.

This is similar to a kernel and is called a "Turing kernel."

Can we “reduce” the k-leaf outbranching problem to the rooted k-leaf outbranching problem?

Yes and No. Depends on what “reduce” exactly means.

We can take a k-leaf outbranching problem and reduce it to n instances of rooted k-leaf outbranching.

This is similar to a kernel and is called a “Turing kernel.”

The Exponential Time Hypothesis

We will use this simple form of the Exponential Time Hypothesis (ETH):

There is a constant $\alpha > 0$ such that no algorithm can solve 3-SAT in at most $2^{\alpha n}(n+m)^{O(1)}$ time.

In particular this implies:

There is no algorithm that solves 3-SAT in $2^{o(n)}(n+m)^{O(1)}$.

The ETH is a complexity theoretic assumption (like $P \neq NP$).

$P \neq NP$ follows from ETH, but not necessarily the other way around.

The Exponential Time Hypothesis

We will use this simple form of the Exponential Time Hypothesis (ETH):

There is a constant $\alpha > 0$ such that no algorithm can solve 3-SAT in at most $2^{\alpha n}(n+m)^{O(1)}$ time.

In particular this implies:

There is no algorithm that solves 3-SAT in $2^{o(n)}(n+m)^{O(1)}$.

The ETH is a complexity theoretic assumption (like $P \neq NP$).

$P \neq NP$ follows from ETH, but not necessarily the other way around.

The Exponential Time Hypothesis

We will use this simple form of the Exponential Time Hypothesis (ETH):

There is a constant $\alpha > 0$ such that no algorithm can solve 3-SAT in at most $2^{\alpha n}(n+m)^{O(1)}$ time.

In particular this implies:

There is no algorithm that solves 3-SAT in $2^{o(n)}(n+m)^{O(1)}$.

The ETH is a complexity theoretic assumption (like $P \neq NP$).

$P \neq NP$ follows from ETH, but not necessarily the other way around.

Subexponential time lower bounds

Assume that we can solve Indepented Set in $2^{o(n)}$ steps (where n is the number of vertices).

How fast can we then solve 3-SAT by reducing it to an IS instance and solve this instance with the above subexponential time solver?

Let ϕ be a 3-SAT instance with n variables m clauses.

We can reduce it to an IS instance with $3m$ vertices.

This can be solved in $2^{o(3m)} = 2^{o(m)}$ time.

It does not contradict the ETH.

Subexponential time lower bounds

Assume that we can solve Indepented Set in $2^{o(n)}$ steps (where n is the number of vertices).

How fast can we then solve 3-SAT by reducing it to an IS instance and solve this instance with the above subexponential time solver?

Let ϕ be a 3-SAT instance with n variables m clauses.

We can reduce it to an IS instance with $3m$ vertices.

This can be solved in $2^{o(3m)} = 2^{o(m)}$ time.

It does not contradict the ETH.

Subexponential time lower bounds

Assume that we can solve Indepented Set in $2^{o(n)}$ steps (where n is the number of vertices).

How fast can we then solve 3-SAT by reducing it to an IS instance and solve this instance with the above subexponential time solver?

Let ϕ be a 3-SAT instance with n variables m clauses.

We can reduce it to an IS instance with $3m$ vertices.

This can be solved in $2^{o(3m)} = 2^{o(m)}$ time.

It does not contradict the ETH.

The Sparsification Lemma

Theorem

For $\epsilon > 0$ and an integer $r > 0$ there is a constant c such that:

1. *For every r-CNF formula ϕ with n variables there is a disjunction ψ of at most $2^{\epsilon n}$ many r-CNF formulas in which every variable occurs in at most c clauses.*
2. *ϕ is satisfiable iff ψ is satisfiable.*
3. *ψ can be computed in $2^{\epsilon n} n^{O(1)}$ time.*

Subexponential time lower bounds

Assume that we can solve Indepented Set in $2^{o(n)}$ steps (where n is the number of vertices).

How fast can we then solve 3-SAT by reducing it to an IS instance and solve this instance with the above subexponential time solver?

Let ϕ be a 3-SAT instance with n variables m clauses.

Use the sparsification lemma to get formulas ψ_i.

The length of each ψ_i is only $O(n)$.

Turn each ψ_i into an IS instance with $O(n)$ vertices.

Solve them in $2^{o(n)}$ steps.

Lower bounds on parameterized problems

If we can reduce 3-SAT to a parameterized problem L such that the parameter is bounded by $O(n+m)$, then L cannot be solved in time $2^{o(k)} poly(n)$ (under ETH).

Proof: Assume otherwise. Then we can solve 3-SAT in time $2^{o(n+m)}$, which contradicts ETH:

First transform a 3-SAT instance I into an L-instance (x, k). Then $|x| = poly(|I|)$ and $k = O(n+m)$. This takes polynomial time.

Then solve $(x, k) \in L$ in time

$$2^{o(k)} poly(|x|) = 2^{o(n+m)} poly(n+m) = 2^{o(n+m)}.$$

Corollary: Vertex Cover and Feedback Vertex Set (and many other problems) cannot be solved in time $2^{o(k)} poly(n)$ under ETH.

Planar 3-SAT

The incidence graph of a 3-SAT formula has a node for each clause and a node for each variable. There is an edge if the variable occurs in a clause.

Planar 3-SAT consists of all satisfiable 3-SAT instances whose incidence graph is planar.

We can reduce in polynomial time a 3-SAT with n variables and m clauses to a Planar 3-SAT instance with $O((n+m)^2)$ clauses and variables.

Proof idea: Replace each crossing by a planar crossover gadget. There are at most nm crossings.

Planar 3-SAT

We cannot solve Planar 3-SAT in time $2^{o(\sqrt{n})}$ under ETH.

Proof: Assume otherwise. Take a 3-SAT instance with n variables and $O(n)$ clauses and transform it into a Planar 3-SAT instance with $O(n^2)$ variables. Then solve it in $2^{(o\sqrt{n^2})} = 2^{o(n)}$ time.

Contradiction.

Corollary: *We cannot solve Planar Vertex Cover and Planar Dominating Set in time $2^{o(\sqrt{k})} poly(n)$ under ETH.*

An artificial problem

The problem $k \times k$-clique:

Input: Graph G with $V(G) = \{1, \ldots, k\}^2$

Parameter: k

Question: Is there a k-clique with one vertex from "each row"?

Lemma

$k \times k$-clique cannot be solved in $2^{o(k \log k)}$ under ETH.

Suppose otherwise. Then we can solve 3-coloring in $2^{o(n)}$.

Proof idea: Given a graph G with n nodes.

Let k be the smallest number with $3^{n/k+1} \leq k$.

(Then $k \log k = O(n)$ and $n/k = O(\log n)$.)

Evenly partition vertices of G into $X_1, \ldots, X_k$.

Construct graph with proper 3-colorings of X_i's as vertices and an edge between "compatible" colorings.

There is a special k-clique iff G is 3-colorable.

Lower bound can be transferred to closest string:

Under ETH closest string cannot be solved in $2^{o(m \log m)} = m^{o(m)}$.