Succinct Compilation of Propositional Theories

Simone Bova

Vienna University of Technology

Universidad del País Vasco
February 6, 2013

Outline

Outline

Classical Compilation

Parameterized Compilation

Research Agenda

Idea

Many reasoning tasks in artificial intelligence (inference, decision) are *computationally intractable*.

Idea

Many reasoning tasks in artificial intelligence (inference, decision) are *computationally intractable*.

Example (LATINCOMPLETION)

1		
	2	

Problem LATINCOMPLETION

Instance A partial function $f : [n] \times [n] \to [n]$.

Idea

Many reasoning tasks in artificial intelligence (inference, decision) are *computationally intractable*.

*Example (*LATINCOMPLETION*)*

1		
	2	

$\rightsquigarrow$

1	3	2
3	2	1
2	1	3

Problem LATINCOMPLETION

Instance A partial function $f : [n] \times [n] \to [n]$.

Question Does there exist a $n \times n$ Latin square extending f?

LATINCOMPLETION is computationally intractable (Colbourn, 1984).

Idea

However, part of the information specifying such tasks is typically *background knowledge*, ie:

Idea

However, part of the information specifying such tasks is typically *background knowledge*, ie:

1. *known before* the execution of individual tasks;

Idea

However, part of the information specifying such tasks is typically *background knowledge*, ie:

1. *known before* the execution of individual tasks;
2. *remains stable* through the execution of several individual tasks.

Idea

A *proposition* is a Boolean formula
(Boolean variables combined by $\neg$, $\wedge$, $\vee$).

Example (Cont'd)

1		
	2	

Idea

A *proposition* is a Boolean formula
(Boolean variables combined by $\neg$, $\wedge$, $\vee$).

Example (Cont'd)

1		
	2	

Is the proposition $\phi_3 \wedge x_{111} \wedge x_{222}$ satisfiable?

Idea

A *proposition* is a Boolean formula
(Boolean variables combined by $\neg$, $\wedge$, $\vee$).

Example (Cont'd)

1		
	2	

$\rightsquigarrow$

1	3	2
3	2	1
2	1	3

Is the proposition $\phi_3 \wedge x_{111} \wedge x_{222}$ satisfiable? Yes!

Idea

A *proposition* is a Boolean formula
(Boolean variables combined by $\neg$, $\wedge$, $\vee$).

Example (Cont'd)

1		
	2	

$\rightsquigarrow$

1	3	2
3	2	1
2	1	3

Is the proposition $\phi_3 \wedge x_{111} \wedge x_{222}$ satisfiable? Yes!

In the above LatinCompletion instance:

1. ϕ_3, the propositional theory of the 3×3 Latin square, is *background knowledge* (known, stable);

Idea

A *proposition* is a Boolean formula
(Boolean variables combined by $\neg$, $\wedge$, $\vee$).

Example (Cont'd)

1		
	2	

$\rightsquigarrow$

1	3	2
3	2	1
2	1	3

Is the proposition $\phi_3 \wedge x_{111} \wedge x_{222}$ satisfiable? Yes!

In the above LATINCOMPLETION instance:

1. ϕ_3, the propositional theory of the 3×3 Latin square, is *background knowledge* (known, stable);
2. $x_{111} \wedge x_{222}$, the given partial function, is *online information* (unknown, varying).

Idea

A *proposition* is a Boolean formula
(Boolean variables combined by $\neg$, $\wedge$, $\vee$).

Example (Cont'd)

$$\begin{array}{|c|c|c|}\hline 1 & & \\ \hline & 2 & \\ \hline & & \\ \hline \end{array} \quad \leadsto \quad \begin{array}{|c|c|c|}\hline 1 & 3 & 2 \\ \hline 3 & 2 & 1 \\ \hline 2 & 1 & 3 \\ \hline \end{array}$$

Is the proposition $\phi_3 \wedge x_{111} \wedge x_{222}$ satisfiable? Yes!

In the above LATINCOMPLETION instance:

1. ϕ_3, the propositional theory of the 3×3 Latin square, is *background knowledge* (known, stable);
2. $x_{111} \wedge x_{222}$, the given partial function, is *online information* (unknown, varying).

Infer the solution by combining *1* and *2*.

Idea

Example (Cont'd)

Idea

Example (Cont'd)

For $(i, j, k) \in [n]^3$, the propositional variable x_{ijk} means, "(i, j) maps to k".

Idea

Example (Cont'd)

For $(i, j, k) \in [n]^3$, the propositional variable x_{ijk} means, "(i, j) maps to k ".

ϕ_n is the *propositional theory* of the $n \times n$ Latin square, ie, satisfying assignments to ϕ_n correspond to $n \times n$ Latin squares (mapping (i, j) to k iff the assignment maps variable x_{ijk} to $\top$).

Idea

Example (Cont'd)

For $(i, j, k) \in [n]^3$, the propositional variable x_{ijk} means, "(i, j) maps to k".

ϕ_n is the *propositional theory* of the $n \times n$ Latin square, ie, satisfying assignments to ϕ_n correspond to $n \times n$ Latin squares (mapping (i, j) to k iff the assignment maps variable x_{ijk} to $\top$).

$\phi_n = \phi_{n1} \wedge \phi_{n2} \wedge \phi_{n3}$ where:

$$\phi_{n1} = \bigwedge_{(i,j)\in[n]^2} \left(\left(\bigvee_{k\in[n]} x_{ijk} \right) \wedge \bigwedge_{k\in[n]} \left(x_{ijk} \to \left(\bigwedge_{k\neq k'\in[n]} \neg x_{ijk'} \right) \right) \right),$$

$$\phi_{n2} = \bigwedge_{(i,k)\in[n]^2} \left(\left(\bigvee_{j\in[n]} x_{ijk} \right) \wedge \bigwedge_{j\in[n]} \left(x_{ijk} \to \left(\bigwedge_{j\neq j'\in[n]} \neg x_{ij'k} \right) \right) \right),$$

$$\phi_{n3} = \bigwedge_{(j,k)\in[n]^2} \left(\left(\bigvee_{i\in[n]} x_{ijk} \right) \wedge \bigwedge_{i\in[n]} \left(x_{ijk} \to \left(\bigwedge_{i\neq i'\in[n]} \neg x_{i'jk} \right) \right) \right).$$

Idea

Exploit *background knowledge against computational intractability*:

Idea

Exploit *background knowledge against computational intractability*:

1. preprocess the background knowledge into a *compiled knowledge* that allows for solving the reasoning task easily (in polynomial time);

Idea

Exploit *background knowledge against computational intractability*:

1. preprocess the background knowledge into a *compiled knowledge* that allows for solving the reasoning task easily (in polynomial time);
2. process *many* individual tasks using the shared compiled knowledge together with task specific online information.

Idea

Exploit *background knowledge against computational intractability*:

1. preprocess the background knowledge into a *compiled knowledge* that allows for solving the reasoning task easily (in polynomial time);
2. process *many* individual tasks using the shared compiled knowledge together with task specific online information.

Compilation cost is amortized by reusing compiled knowledge
to ease a large number of individual executions.

Entailment

The key problem in knowledge compilation since the 90s:

Problem CLAUSEENTAILMENT

Instance A proposition ϕ (*theory*) and a clause δ (*query*).

Question $\phi \models \delta$?

Entailment

The key problem in knowledge compilation since the 90s:

Problem ClauseEntailment

Instance A proposition ϕ (*theory*) and a clause δ (*query*).

Question $\phi \models \delta$?

				ϕ	δ_1	δ_2
w	x	y	z	$(x \vee z) \wedge (x \vee y) \wedge (\neg w \vee y \vee \neg z) \wedge (\neg w \vee \neg x \vee \neg y)$	$\neg w \vee \neg y \vee z$	$y \vee z$
0	0	0	0	0	1	0
0	0	0	1	0	1	1
0	0	1	0	0	1	1
0	**0**	**1**	**1**	**1**	**1**	**1**
0	**1**	**0**	**0**	**1**	**1**	**0**
0	**1**	**0**	**1**	**1**	**1**	**1**
0	**1**	**1**	**0**	**1**	**1**	**1**
0	**1**	**1**	**1**	**1**	**1**	**1**
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	0	0	1
1	**0**	**1**	**1**	**1**	**1**	**1**
1	**1**	**0**	**0**	**1**	**1**	**0**
1	1	0	1	0	1	1
1	1	1	0	0	0	1
1	1	1	1	0	1	1

Entailment

CLAUSEENTAILMENT is *computationally intractable* (coNP-hard).

Entailment

ClauseEntailment is *computationally intractable* (coNP-hard).

Take ϕ, the theory, as *background knowledge*
and δ, the query, as *online information*
(practical case in artificial intelligence).

Entailment

CLAUSEENTAILMENT is *computationally intractable* (coNP-hard).

Take ϕ, the theory, as *background knowledge*
and δ, the query, as *online information*
(practical case in artificial intelligence).

Definition (Compilation)

A *compilation* is a (computable) map c st for all ϕ and δ:

1. $c(\phi) \models \delta$ iff $\phi \models \delta$ (ie, $c(\phi)$ logically equivalent to ϕ);
2. $c(\phi) \models \delta$ is poly-time decidable.

Entailment

CLAUSEENTAILMENT is *computationally intractable* (coNP-hard).

Take ϕ, the theory, as *background knowledge*
and δ, the query, as *online information*
(practical case in artificial intelligence).

Definition (Compilation)

A *compilation* is a (computable) map c st for all ϕ and δ:

1. $c(\phi) \models \delta$ iff $\phi \models \delta$ (ie, $c(\phi)$ logically equivalent to ϕ);
2. $c(\phi) \models \delta$ is poly-time decidable.

A series of *hard* instances,

$$\begin{array}{c}(\phi, \delta_1)\\(\phi, \delta_2)\\(\phi, \delta_3)\\\vdots\end{array}$$

Entailment

CLAUSEENTAILMENT is *computationally intractable* (coNP-hard).

Take ϕ, the theory, as *background knowledge*
and δ, the query, as *online information*
(practical case in artificial intelligence).

Definition (Compilation)

A *compilation* is a (computable) map c st for all ϕ and δ:

1. $c(\phi) \models \delta$ iff $\phi \models \delta$ (ie, $c(\phi)$ logically equivalent to ϕ);
2. $c(\phi) \models \delta$ is poly-time decidable.

A series of *hard* instances, *compiles* into a series of *easy equivalent* instances:

$$\begin{array}{ccc} (\phi, \delta_1) & \rightsquigarrow & (c(\phi), \delta_1) \\ (\phi, \delta_2) & \rightsquigarrow & (c(\phi), \delta_2) \\ (\phi, \delta_3) & \rightsquigarrow & (c(\phi), \delta_3) \\ \vdots & \vdots & \vdots \end{array}$$

Entailment

Example (Compilation into DNF)

Compile ϕ into DNF $c(\phi)$ logically equivalent to ϕ, eg:

Entailment

Example (Compilation into DNF)

Compile ϕ into DNF $c(\phi)$ logically equivalent to ϕ, eg:

$$\phi = (x_1 \vee x_2) \wedge (x_3 \vee x_4),$$

Entailment

Example (Compilation into DNF)

Compile ϕ into DNF $c(\phi)$ logically equivalent to ϕ, eg:

$$\phi = (x_1 \vee x_2) \wedge (x_3 \vee x_4),$$
$$c(\phi) = (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4).$$

Entailment

Example (Compilation into DNF)

Compile ϕ into DNF $c(\phi)$ logically equivalent to ϕ, eg:

$$\begin{aligned} \phi &= (x_1 \vee x_2) \wedge (x_3 \vee x_4), \\ c(\phi) &= (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4). \end{aligned}$$

Check $c(\phi) \models \delta$, eg, $\delta = \neg x_3 \vee x_4$:

Entailment

Example (Compilation into DNF)

Compile ϕ into DNF $c(\phi)$ logically equivalent to ϕ, eg:

$$\begin{aligned} \phi &= (x_1 \vee x_2) \wedge (x_3 \vee x_4), \\ c(\phi) &= (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4). \end{aligned}$$

Check $c(\phi) \models \delta$, eg, $\delta = \neg x_3 \vee x_4$:

$c(\phi) \models \delta$ iff $c(\phi) \wedge \neg\delta$ unsatisfiable,

Entailment

Example (Compilation into DNF)

Compile ϕ into DNF $c(\phi)$ logically equivalent to ϕ, eg:

$$\begin{aligned} \phi &= (x_1 \vee x_2) \wedge (x_3 \vee x_4), \\ c(\phi) &= (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4). \end{aligned}$$

Check $c(\phi) \models \delta$, eg, $\delta = \neg x_3 \vee x_4$:

$c(\phi) \models \delta$ iff $c(\phi) \wedge \neg\delta$ unsatisfiable,

iff $c(\phi) \wedge (x_3 \wedge \neg x_4)$ unsatisfiable,

Entailment

Example (Compilation into DNF)

Compile ϕ into DNF $c(\phi)$ logically equivalent to ϕ, eg:

$$\begin{aligned} \phi &= (x_1 \vee x_2) \wedge (x_3 \vee x_4), \\ c(\phi) &= (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4). \end{aligned}$$

Check $c(\phi) \models \delta$, eg, $\delta = \neg x_3 \vee x_4$:

$$\begin{aligned} c(\phi) \models \delta \text{ iff } & c(\phi) \wedge \neg\delta \text{ unsatisfiable,} \\ \text{iff } & c(\phi) \wedge (x_3 \wedge \neg x_4) \text{ unsatisfiable,} \\ \text{iff } & ((x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4)) \wedge (x_3 \wedge \neg x_4) \text{ unsatisfiable,} \end{aligned}$$

Entailment

Example (Compilation into DNF)

Compile ϕ into DNF $c(\phi)$ logically equivalent to ϕ, eg:

$$\begin{aligned} \phi &= (x_1 \vee x_2) \wedge (x_3 \vee x_4), \\ c(\phi) &= (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4). \end{aligned}$$

Check $c(\phi) \models \delta$, eg, $\delta = \neg x_3 \vee x_4$:

$$\begin{aligned} c(\phi) \models \delta \text{ iff } & c(\phi) \wedge \neg\delta \text{ unsatisfiable,} \\ \text{iff } & c(\phi) \wedge (x_3 \wedge \neg x_4) \text{ unsatisfiable,} \\ \text{iff } & ((x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4)) \wedge (x_3 \wedge \neg x_4) \text{ unsatisfiable,} \\ \text{iff } & x_1 \vee x_2 \text{ unsatisfiable (false).} \end{aligned}$$

Entailment

Example (Compilation into DNF)

Compile ϕ into DNF $c(\phi)$ logically equivalent to ϕ, eg:

$$\begin{aligned} \phi &= (x_1 \vee x_2) \wedge (x_3 \vee x_4), \\ c(\phi) &= (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4). \end{aligned}$$

Check $c(\phi) \models \delta$, eg, $\delta = \neg x_3 \vee x_4$:

$$\begin{aligned} c(\phi) \models \delta &\text{ iff } c(\phi) \wedge \neg\delta \text{ unsatisfiable}, \\ &\text{ iff } c(\phi) \wedge (x_3 \wedge \neg x_4) \text{ unsatisfiable}, \\ &\text{ iff } ((x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4)) \wedge (x_3 \wedge \neg x_4) \text{ unsatisfiable}, \\ &\text{ iff } x_1 \vee x_2 \text{ unsatisfiable (false)}. \end{aligned}$$

Thus, ClauseEntailment compiles via such c:

Entailment

Example (Compilation into DNF)

Compile ϕ into DNF $c(\phi)$ logically equivalent to ϕ, eg:

$$\phi = (x_1 \vee x_2) \wedge (x_3 \vee x_4),$$
$$c(\phi) = (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4).$$

Check $c(\phi) \models \delta$, eg, $\delta = \neg x_3 \vee x_4$:

$c(\phi) \models \delta$ iff $c(\phi) \wedge \neg\delta$ unsatisfiable,

iff $c(\phi) \wedge (x_3 \wedge \neg x_4)$ unsatisfiable,

iff $((x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4)) \wedge (x_3 \wedge \neg x_4)$ unsatisfiable,

iff $x_1 \vee x_2$ unsatisfiable (false).

Thus, CLAUSEENTAILMENT compiles via such c:

1. $c(\phi) \models \delta$ iff $\phi \models \delta$ for all δ;

Entailment

Example (Compilation into DNF)

Compile ϕ into DNF $c(\phi)$ logically equivalent to ϕ, eg:

$$\phi = (x_1 \vee x_2) \wedge (x_3 \vee x_4),$$
$$c(\phi) = (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4).$$

Check $c(\phi) \models \delta$, eg, $\delta = \neg x_3 \vee x_4$:

$c(\phi) \models \delta$ iff $c(\phi) \wedge \neg\delta$ unsatisfiable,

iff $c(\phi) \wedge (x_3 \wedge \neg x_4)$ unsatisfiable,

iff $((x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4)) \wedge (x_3 \wedge \neg x_4)$ unsatisfiable,

iff $x_1 \vee x_2$ unsatisfiable (false).

Thus, CLAUSEENTAILMENT compiles via such c:

1. $c(\phi) \models \delta$ iff $\phi \models \delta$ for all δ;
2. $c(\phi) \models \delta$ is poly-time decidable (reduction to DNF satisfiability, easy).

Succinctness

Example (Compilation into DNF, Cont'd)

- $\phi = (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{n-1} \vee x_n)$ is size $|\phi| = n$;

Succinctness

Example (Compilation into DNF, Cont'd)

- $\phi = (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{n-1} \vee x_n)$ is size $|\phi| = n$;
- $|c(\phi)| \geq |(x_1 \wedge x_3 \wedge \cdots \wedge x_{n-1}) \vee \cdots \vee (x_2 \wedge x_4 \wedge \cdots \wedge x_n)| \geq 2^{n/2} \cdot n/2$;

Succinctness

Example (Compilation into DNF, Cont'd)

- $\phi = (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{n-1} \vee x_n)$ is size $|\phi| = n$;
- $|c(\phi)| \geq |(x_1 \wedge x_3 \wedge \cdots \wedge x_{n-1}) \vee \cdots \vee (x_2 \wedge x_4 \wedge \cdots \wedge x_n)| \geq 2^{n/2} \cdot n/2$;
- $|c(\phi)|$ is *not polynomially bounded* in the size of $|\phi|$.

Succinctness

Example (Compilation into DNF, Cont'd)

- $\phi = (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{n-1} \vee x_n)$ is size $|\phi| = n$;
- $|c(\phi)| \geq |(x_1 \wedge x_3 \wedge \cdots \wedge x_{n-1}) \vee \cdots \vee (x_2 \wedge x_4 \wedge \cdots \wedge x_n)| \geq 2^{n/2} \cdot n/2$;
- $|c(\phi)|$ is *not polynomially bounded* in the size of $|\phi|$.

Definition (Succinctness)

A compilation c is *succinct* if $|c(\phi)|$ is polynomially bounded in $|\phi|$, ie, there exists d st for all ϕ,

$$|c(\phi)| \in O(|\phi|^d).$$

Succinctness

Example (Compilation into DNF, Cont'd)

- $\phi = (x_1 \vee x_2) \wedge (x_3 \vee x_4) \wedge \cdots \wedge (x_{n-1} \vee x_n)$ is size $|\phi| = n$;
- $|c(\phi)| \geq |(x_1 \wedge x_3 \wedge \cdots \wedge x_{n-1}) \vee \cdots \vee (x_2 \wedge x_4 \wedge \cdots \wedge x_n)| \geq 2^{n/2} \cdot n/2$;
- $|c(\phi)|$ is *not polynomially bounded* in the size of $|\phi|$.

Definition (Succinctness)

A compilation c is *succinct* if $|c(\phi)|$ is polynomially bounded in $|\phi|$, ie, there exists d st for all ϕ,

$$|c(\phi)| \in O(|\phi|^d).$$

Remark

Without succinctness, CLAUSEENTAILMENT *compiles even requiring that $c(\phi) \models \delta$ is decidable in time $O(|\phi|^d)$.*

Compilability

LITERALENTAILMENT is CLAUSEENTAILMENT restricted to instances (ϕ, δ) where δ is a literal.

Compilability

LITERALENTAILMENT is CLAUSEENTAILMENT restricted to instances (ϕ, δ) where δ is a literal.

Fact

LITERALENTAILMENT *compiles succinctly.*

Compilability

LITERALENTAILMENT is CLAUSEENTAILMENT restricted to instances (ϕ, δ) where δ is a literal.

Fact

LITERALENTAILMENT *compiles succinctly.*

Proof.

The map c sends ϕ to $c(\phi)$, the conjunction of all literals entailed by ϕ (computing c involves solving $\leq |\phi|$ many instances of a coNP-hard problem). For all literals δ, clearly $c(\phi) \models \delta$ is poly-time decidable (check δ occurs in $c(\phi)$ as a conjunct), $c(\phi) \models \delta$ iff $\phi \models \delta$. Moreover, $|c(\phi)| \leq |\phi|$, thus c is succinct. □

Classical Compilability | Incompilability

Theorem (Selman and Kautz, 1996)

CLAUSEENTAILMENT *does not compile succinctly (under standard assumptions in complexity theory).*

Classical Compilability | Incompilability

Theorem (Selman and Kautz, 1996)

CLAUSEENTAILMENT *does not compile succinctly (under standard assumptions in complexity theory).*

Proof.

Suppose not. Let $n \in \mathbb{N}$.
Key observation (easy). There exists a proposition τ_n *of size* $O(n^3)$ *st for all 3CNF* χ *on* n *variables, there exists a clause* δ_χ *st* $\tau_n \models \delta_\chi$ *if and only if* χ *is unsatisfiable.*
Let $\tau_n \rightsquigarrow c(\tau_n)$ *be a succint compilation of* τ_n.
We give a polynomial-time algorithm for the satisfiability of 3CNFs on n *variables, ie,* 3SAT *in P/poly which implies NP⊆P/poly and thus PH collapses to* Σ_2^p *(Karp and Lipton, 1980).*
The algorithm, given a propositional formula χ *on* n *variables, decides in polynomial-time the question* $c(\tau_n) \models \delta_\chi$ *(here* $c(\tau_n)$ *is the advice), and reports that* χ *is satisfiable if and only if the answer is negative.* $\square$

Outline

Fixed-Parameter Tractability

3SAT: Given a 3CNF ϕ on n variables, is ϕ satisfiable?

Fixed-Parameter Tractability

3SAT: Given a 3CNF ϕ on n variables, is ϕ satisfiable?

3SAT is NP-hard:

Fixed-Parameter Tractability

3SAT: Given a 3CNF ϕ on n variables, is ϕ satisfiable?

3SAT is NP-hard:

1. solvable in exponential time $O(d^n)$ with $d < 2$;

Fixed-Parameter Tractability

3SAT: Given a 3CNF ϕ on n variables, is ϕ satisfiable?

3SAT is NP-hard:

1. solvable in exponential time $O(d^n)$ with $d < 2$;
2. believed not solvable in subexponential time $2^{o(n)}$.

Fixed-Parameter Tractability

3SAT: Given a 3CNF ϕ on n variables, is ϕ satisfiable?

3SAT is NP-hard:

1. solvable in exponential time $O(d^n)$ with $d < 2$;
2. believed not solvable in subexponential time $2^{o(n)}$.

Theorem

3SAT *is is solvable in time* $O(k2^k \cdot n)$ *where k is the treewidth of the instance*

Fixed-Parameter Tractability

3SAT: Given a 3CNF ϕ on n variables, is ϕ satisfiable?

3SAT is NP-hard:

1. solvable in exponential time $O(d^n)$ with $d < 2$;
2. believed not solvable in subexponential time $2^{o(n)}$.

Theorem
3SAT *is is solvable in time* $O(k2^k \cdot n)$ *where* k *is the treewidth of the instance*

$O(k2^k \cdot n)$ faster than $O(d^n)$ if k is much smaller than n ($k \ll n$).

Fixed-Parameter Tractability

3SAT: Given a 3CNF ϕ on n variables, is ϕ satisfiable?

3SAT is NP-hard:

1. solvable in exponential time $O(d^n)$ with $d < 2$;
2. believed not solvable in subexponential time $2^{o(n)}$.

Theorem
3SAT *is is solvable in time* $O(k2^k \cdot n)$ *where k is the treewidth of the instance*

$O(k2^k \cdot n)$ faster than $O(d^n)$ if k is much smaller than n ($k \ll n$).

Example
Treewidth $\mathrm{tw}(\phi)$ of typical industrial instance ϕ on 2000 vars is < 10.

Fixed-Parameter Tractability

3SAT: Given a 3CNF ϕ on n variables, is ϕ satisfiable?

3SAT is NP-hard:

1. solvable in exponential time $O(d^n)$ with $d < 2$;
2. believed not solvable in subexponential time $2^{o(n)}$.

Theorem

3SAT *is is solvable in time* $O(k2^k \cdot n)$ *where k is the treewidth of the instance, ie,*
3SAT *is fixed-parameter tractable wrt parameterization* tw*, ie,*
it has a runtime of the form $f(\mathrm{tw}(\phi))|\phi|^d$ *for some constant d and function f.*

$O(k2^k \cdot n)$ faster than $O(d^n)$ if k is much smaller than n ($k \ll n$).

Example

Treewidth $\mathrm{tw}(\phi)$ of typical industrial instance ϕ on 2000 vars is < 10.

Treewidth

Example

$\phi = (\neg x_7 \vee \neg x_5 \vee \neg x_3) \wedge (x_4 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_8 \vee \neg x_4) \wedge (\neg x_8 \vee x_6 \vee \neg x_5) \wedge (x_4 \vee \neg x_1 \vee \neg x_7)$.

Treewidth

Example

$\phi = (\neg x_7 \vee \neg x_5 \vee \neg x_3) \wedge (x_4 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_8 \vee \neg x_4) \wedge (\neg x_8 \vee x_6 \vee \neg x_5) \wedge (x_4 \vee \neg x_1 \vee \neg x_7)$.

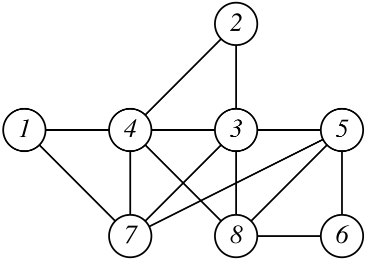

Figure: Primal graph of ϕ.

Treewidth

Example

$\phi = (\neg x_7 \vee \neg x_5 \vee \neg x_3) \wedge (x_4 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_8 \vee \neg x_4) \wedge (\neg x_8 \vee x_6 \vee \neg x_5) \wedge (x_4 \vee \neg x_1 \vee \neg x_7)$.

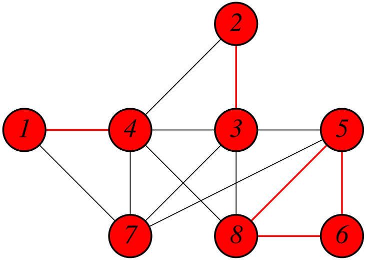

Figure: $\{\{1,4\},\{2,3\},\{5,6,8\},\{7\}\}$ 4-bramble implies $\mathrm{tw}(\phi) \geq 3$.

Treewidth

Example

$\phi = (\neg x_7 \vee \neg x_5 \vee \neg x_3) \wedge (x_4 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_8 \vee \neg x_4) \wedge (\neg x_8 \vee x_6 \vee \neg x_5) \wedge (x_4 \vee \neg x_1 \vee \neg x_7)$.

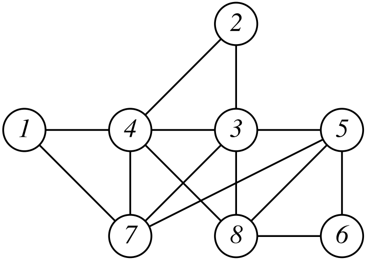

Figure: Primal graph of ϕ. Elimination $2, 1, 6, 5, 4, 3, 8, 7$ gives $\mathrm{tw}(\phi) \leq 3$.

Treewidth

Example

$\phi = (\neg x_7 \vee \neg x_5 \vee \neg x_3) \wedge (x_4 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_8 \vee \neg x_4) \wedge (\neg x_8 \vee x_6 \vee \neg x_5) \wedge (x_4 \vee \neg x_1 \vee \neg x_7)$.

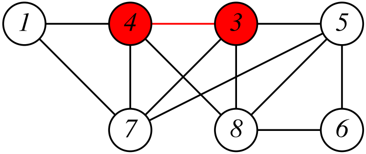

Figure: Eliminating 2, neigborhood size $|\{3, 4\}| = 2 \ldots$

Treewidth

Example

$\phi = (\neg x_7 \vee \neg x_5 \vee \neg x_3) \wedge (x_4 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_8 \vee \neg x_4) \wedge (\neg x_8 \vee x_6 \vee \neg x_5) \wedge (x_4 \vee \neg x_1 \vee \neg x_7)$.

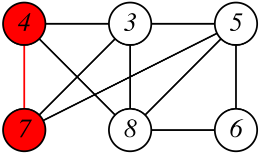

Figure: Eliminating 1, neigborhood size $|\{4, 7\}| = 2 \ldots$

Treewidth

Example

$\phi = (\neg x_7 \vee \neg x_5 \vee \neg x_3) \wedge (x_4 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_8 \vee \neg x_4) \wedge (\neg x_8 \vee x_6 \vee \neg x_5) \wedge (x_4 \vee \neg x_1 \vee \neg x_7)$.

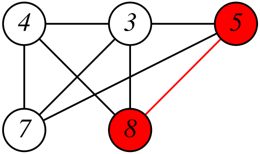

Figure: Eliminating 6, neigborhood size $|\{5, 8\}| = 2 \ldots$

Treewidth

Example

$\phi = (\neg x_7 \vee \neg x_5 \vee \neg x_3) \wedge (x_4 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_8 \vee \neg x_4) \wedge (\neg x_8 \vee x_6 \vee \neg x_5) \wedge (x_4 \vee \neg x_1 \vee \neg x_7)$.

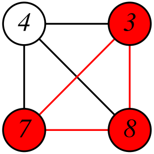

Figure: Eliminating 5, neigborhood size $|\{3, 7, 8\}| = 3 \ldots$

Treewidth

Example

$\phi = (\neg x_7 \vee \neg x_5 \vee \neg x_3) \wedge (x_4 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_8 \vee \neg x_4) \wedge (\neg x_8 \vee x_6 \vee \neg x_5) \wedge (x_4 \vee \neg x_1 \vee \neg x_7)$.

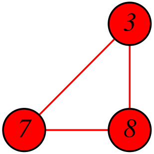

Figure: Eliminating 4, neigborhood size $|\{3, 7, 8\}| = 3$.

Treewidth

Example

$\phi = (\neg x_7 \vee \neg x_5 \vee \neg x_3) \wedge (x_4 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_8 \vee \neg x_4) \wedge (\neg x_8 \vee x_6 \vee \neg x_5) \wedge (x_4 \vee \neg x_1 \vee \neg x_7)$.

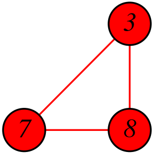

Figure: Eliminating 4, neigborhood size $|\{3, 7, 8\}| = 3$. Done.

Treewidth

Example

$\phi = (\neg x_7 \vee \neg x_5 \vee \neg x_3) \wedge (x_4 \vee x_2 \vee \neg x_3) \wedge (\neg x_3 \vee \neg x_8 \vee \neg x_4) \wedge (\neg x_8 \vee x_6 \vee \neg x_5) \wedge (x_4 \vee \neg x_1 \vee \neg x_7)$.

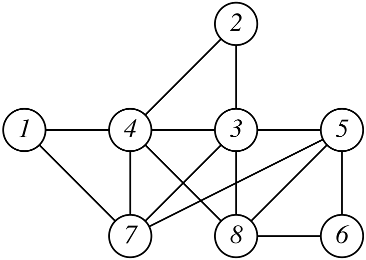

Figure: $\mathrm{tw}(\phi) = 3$.

Parameterized Compilation

ClauseEntailment: Given (ϕ, δ), does $\phi \models \delta$?

Parameterized Compilation

ClauseEntailment: Given (ϕ, δ), does $\phi \models \delta$?

A *parameterization* is a map κ sending pairs (ϕ, δ) into $\mathbb{N}$.

Parameterized Compilation

ClauseEntailment: Given (ϕ, δ), does $\phi \models \delta$?

A *parameterization* is a map κ sending pairs (ϕ, δ) into $\mathbb{N}$.

Definition (Parametrically Succinct Compilation)

Let κ be a parameterization. A compilation c is (wrt parameterization κ):

Parameterized Compilation

CLAUSEENTAILMENT: Given (ϕ, δ), does $\phi \models \delta$?

A *parameterization* is a map κ sending pairs (ϕ, δ) into $\mathbb{N}$.

Definition (Parametrically Succinct Compilation)

Let κ be a parameterization. A compilation c is (wrt parameterization κ):

1. *kernel-size* if $|c(\phi)| \leq f(\kappa(\phi, \delta))$ for some function f;

Parameterized Compilation

ClauseEntailment: Given (ϕ, δ), does $\phi \models \delta$?

A *parameterization* is a map κ sending pairs (ϕ, δ) into $\mathbb{N}$.

Definition (Parametrically Succinct Compilation)

Let κ be a parameterization. A compilation c is (wrt parameterization κ):

1. *kernel-size* if $|c(\phi)| \leq f(\kappa(\phi, \delta))$ for some function f;
2. *fpt-size* (or *fixed-parameter tractable in size*) if $|c(\phi)| \leq f(\kappa(\phi, \delta)) \cdot |(\phi, \delta)|^d$ for some function f and constant d.

Parameterized Compilation

CLAUSEENTAILMENT fails classical compilation, ie, does not compile succinctly (unless PH collapses).

Parameterized Compilation

CLAUSEENTAILMENT fails classical compilation, ie,
does not compile succinctly (unless PH collapses).

Can we relativize *classical incompilability* by *parametrized compilability*? Ie:

Parameterized Compilation

CLAUSEENTAILMENT fails classical compilation, ie,
does not compile succinctly (unless PH collapses).

Can we relativize *classical incompilability* by *parametrized compilability*? Ie:

1. find parameterizations κ st CLAUSEENTAILMENT compiles in kernel-size (wrt κ);

Parameterized Compilation

ClauseEntailment fails classical compilation, ie,
does not compile succinctly (unless PH collapses).

Can we relativize *classical incompilability* by *parametrized compilability*? Ie:

1. find parameterizations κ st ClauseEntailment compiles in kernel-size (wrt κ);
2. find parameterizations κ st ClauseEntailment compiles in fpt-size (wrt κ).

Parameterized Compilation

CLAUSEENTAILMENT fails classical compilation, ie,
does not compile succinctly (unless PH collapses).

Can we relativize *classical incompilability* by *parametrized compilability*? Ie:

1. find parameterizations κ st CLAUSEENTAILMENT compiles in kernel-size (wrt κ);
2. find parameterizations κ st CLAUSEENTAILMENT compiles in fpt-size (wrt κ).

Remark

1. *There are examples witnessing (1) kernel-size compilability, (2 and not 1) fpt-size compilability but kernel-size incompilability, and (not 2) fpt-size incompilability.*

Parameterized Compilation

CLAUSEENTAILMENT fails classical compilation, ie,
does not compile succinctly (unless PH collapses).

Can we relativize *classical incompilability* by *parametrized compilability*? Ie:

1. find parameterizations κ st CLAUSEENTAILMENT compiles in kernel-size (wrt κ);
2. find parameterizations κ st CLAUSEENTAILMENT compiles in fpt-size (wrt κ).

Remark

1. *There are examples witnessing (1) kernel-size compilability, (2 and not 1) fpt-size compilability but kernel-size incompilability, and (not 2) fpt-size incompilability.*
2. *Parameterizations κ yielding fixed-parameter tractability of* CLAUSEENTAILMENT *are uninteresting wrt parameterized compilation.*

Implicates

ϕ is a proposition, δ is a clause:

Implicates

ϕ is a proposition, δ is a clause:

1. δ *implicate* of ϕ if $\phi \models \delta$ and $\top \not\models \delta$;

Implicates

ϕ is a proposition, δ is a clause:

1. δ *implicate* of ϕ if $\phi \models \delta$ and $\top \not\models \delta$;
2. δ *prime* implicate of ϕ if,
   $\phi \models \delta' \models \delta$ implies $\delta \models \delta'$ for all implicates δ' of ϕ.

Implicates

ϕ is a proposition, δ is a clause:

1. δ *implicate* of ϕ if $\phi \models \delta$ and $\top \not\models \delta$;
2. δ *prime* implicate of ϕ if,
   $\phi \models \delta' \models \delta$ implies $\delta \models \delta'$ for all implicates δ' of ϕ.

pif(ϕ), *prime implicate form* of ϕ, is conjunction of prime implicates of ϕ.

Implicates

ϕ is a proposition, δ is a clause:

1. δ *implicate* of ϕ if $\phi \models \delta$ and $\top \not\models \delta$;
2. δ *prime* implicate of ϕ if,
   $\phi \models \delta' \models \delta$ implies $\delta \models \delta'$ for all implicates δ' of ϕ.

$\mathrm{pif}(\phi)$, *prime implicate form* of ϕ, is conjunction of prime implicates of ϕ.

Fact

1. *For all clauses δ, $\phi \models \delta$ iff $\delta_i \models \delta$ for some clause δ_i of* $\mathrm{pif}(\phi)$.

Implicates

ϕ is a proposition, δ is a clause:

1. δ *implicate* of ϕ if $\phi \models \delta$ and $\top \not\models \delta$;
2. δ *prime* implicate of ϕ if,
   $\phi \models \delta' \models \delta$ implies $\delta \models \delta'$ for all implicates δ' of ϕ.

$\mathrm{pif}(\phi)$, *prime implicate form* of ϕ, is conjunction of prime implicates of ϕ.

Fact

1. *For all clauses* δ, $\phi \models \delta$ *iff* $\delta_i \models \delta$ *for some clause* δ_i *of* $\mathrm{pif}(\phi)$.
2. $\mathrm{pif}(\phi) \models \delta$ *is poly-time.*

Implicates

ϕ is a proposition, δ is a clause:

1. δ *implicate* of ϕ if $\phi \models \delta$ and $\top \not\models \delta$;
2. δ *prime* implicate of ϕ if,
   $\phi \models \delta' \models \delta$ implies $\delta \models \delta'$ for all implicates δ' of ϕ.

$\mathrm{pif}(\phi)$, *prime implicate form* of ϕ, is conjunction of prime implicates of ϕ.

Fact

1. *For all clauses δ, $\phi \models \delta$ iff $\delta_i \models \delta$ for some clause δ_i of* $\mathrm{pif}(\phi)$.
2. $\mathrm{pif}(\phi) \models \delta$ *is poly-time.*
3. $\mathrm{pif}(\phi)$ *is logically equivalent to ϕ.*

Implicates

ϕ is a proposition, δ is a clause:

1. δ *implicate* of ϕ if $\phi \models \delta$ and $\top \not\models \delta$;
2. δ *prime* implicate of ϕ if,
   $\phi \models \delta' \models \delta$ implies $\delta \models \delta'$ for all implicates δ' of ϕ.

$\mathrm{pif}(\phi)$, *prime implicate form* of ϕ, is conjunction of prime implicates of ϕ.

Fact

1. *For all clauses δ, $\phi \models \delta$ iff $\delta_i \models \delta$ for some clause δ_i of* $\mathrm{pif}(\phi)$.
2. $\mathrm{pif}(\phi) \models \delta$ *is poly-time.*
3. $\mathrm{pif}(\phi)$ *is logically equivalent to ϕ.*

Remark
Prime implicate forms can be redundant.
Irredundant prime implicate forms are not unique.

Implicates

				1	✓	✓	✓			✓
				2	✓	✓		✓	✓	
				3	✓	✓			✓	✓
w	x	y	z	ϕ	$x \vee z$	$x \vee y$	$\neg w \vee y \vee \neg z$	$\neg w \vee \neg y \vee z$	$\neg w \vee \neg x \vee \neg z$	$\neg w \vee \neg x \vee \neg y$
0	0	0	0	0	0	0	1	1	1	1
0	0	0	1	0	1	0	1	1	1	1
0	0	1	0	0	0	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1
0	1	0	0	1	1	1	1	1	1	1
0	1	0	1	1	1	1	1	1	1	1
0	1	1	0	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1
1	0	0	1	0	1	0	0	1	1	1
1	0	1	0	0	0	1	1	0	1	1
1	0	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1
1	1	0	1	0	1	1	0	1	0	1
1	1	1	0	0	1	1	1	0	1	0
1	1	1	1	0	1	1	1	1	0	0

ϕ has 3 irredundant prime implicate forms.

Kernel-Size Compilation

Parameterization $\mathrm{minvar}(\phi, \delta)$ is the smallest $k \in \mathbb{N}$ such that ϕ is logically equivalent to a proposition on k variables.

Kernel-Size Compilation

Parameterization minvar(ϕ, δ) is the smallest $k \in \mathbb{N}$ such that ϕ is logically equivalent to a proposition on k variables.

Observation
ClauseEntailment *compiles in kernel-size wrt parameterization* minvar.

Kernel-Size Compilation

Parameterization $\operatorname{minvar}(\phi, \delta)$ is the smallest $k \in \mathbb{N}$ such that
ϕ is logically equivalent to a proposition on k variables.

Observation
CLAUSEENTAILMENT *compiles in kernel-size wrt parameterization* minvar.

Proof.
Let ϕ be a proposition. Take $c(\phi)$ be the prime implicate normal form of ϕ (computable by Quine and McKluskey algorithm, hard).
Then $c(\phi)$ uses exactly $\operatorname{minvar}(\phi, \delta) = k$ variables, thus $|c(\phi)| \leq k2^k$. $\square$

Kernel-Size Compilation

Parameterization $\mathrm{minvar}(\phi, \delta)$ is the smallest $k \in \mathbb{N}$ such that ϕ is logically equivalent to a proposition on k variables.

Observation

CLAUSEENTAILMENT *compiles in kernel-size wrt parameterization* minvar.

Proof.

Let ϕ be a proposition. Take $c(\phi)$ be the prime implicate normal form of ϕ (computable by Quine and McKluskey algorithm, hard).
Then $c(\phi)$ uses exactly $\mathrm{minvar}(\phi, \delta) = k$ *variables, thus* $|c(\phi)| \leq k2^k$. □

Conjecture

CLAUSEENTAILMENT *not in fpt-time wrt parameterization* minvar.

Kernel-Size Compilation

$\mathcal{F}$ class of propositions, κ parameterization.
$\mathcal{F}$ is *κ-bounded* if there exists k st for all $\phi \in \mathcal{F}$, $\kappa(\phi) \leq k$.

Kernel-Size Compilation

$\mathcal{F}$ class of propositions, κ parameterization.
$\mathcal{F}$ is *κ-bounded* if there exists k st for all $\phi \in \mathcal{F}$, $\kappa(\phi) \leq k$.

CLAUSEENTAILMENT($\mathcal{F}$) is CLAUSEENTAILMENT restricted to instances (ϕ, δ) with $\phi \in \mathcal{F}$.

Kernel-Size Compilation

$\mathcal{F}$ class of propositions, κ parameterization.
$\mathcal{F}$ is κ*-bounded* if there exists k st for all $\phi \in \mathcal{F}$, $\kappa(\phi) \leq k$.

CLAUSEENTAILMENT($\mathcal{F}$) is CLAUSEENTAILMENT
restricted to instances (ϕ, δ) with $\phi \in \mathcal{F}$.

Conjecture

CLAUSEENTAILMENT($\mathcal{F}$) *compiles in constant-size if and only if* $\mathcal{F}$ *is* minvar-*bounded.*

Kernel-Size Compilation

$\mathcal{F}$ class of propositions, κ parameterization.
$\mathcal{F}$ is *κ-bounded* if there exists k st for all $\phi \in \mathcal{F}$, $\kappa(\phi) \leq k$.

CLAUSEENTAILMENT($\mathcal{F}$) is CLAUSEENTAILMENT restricted to instances (ϕ, δ) with $\phi \in \mathcal{F}$.

Conjecture

CLAUSEENTAILMENT($\mathcal{F}$) *compiles in constant-size if and only if $\mathcal{F}$ is* minvar-*bounded.*

The proposition gives sufficiency (necessity is open).

Fpt-Size Compilation

Parameterization $\mathrm{mintw}(\phi, \delta)$ is the smallest $k \in \mathbb{N}$ such that ϕ is logically equivalent to a CNF of treewidth k.

Fpt-Size Compilation

Parameterization mintw(ϕ, δ) is the smallest $k \in \mathbb{N}$ such that ϕ is logically equivalent to a CNF of treewidth k.

Observation
CLAUSEENTAILMENT *compiles in fpt-size wrt parameterization* mintw.

Fpt-Size Compilation

Parameterization $\text{mintw}(\phi, \delta)$ is the smallest $k \in \mathbb{N}$ such that ϕ is logically equivalent to a CNF of treewidth k.

Observation

CLAUSEENTAILMENT *compiles in fpt-size wrt parameterization* mintw.

Proof.

Let ϕ be a proposition using n variables. Let ϕ' be an irredundant prime implicate normal form of ϕ with minimum treewidth (among all irredundant prime implicate normal forms of ϕ). Then, $\text{tw}(\phi') = \text{mintw}(\phi, \delta) = k$. *Take $c(\phi)$ to be the join tree form (a certain CNF) of a small tree decomposition of ϕ' (computable, hard). Then $|c(\phi)| \leq k2^k \cdot n$.* □

Fpt-Size Compilation

Parameterization $\mathrm{mintw}(\phi, \delta)$ is the smallest $k \in \mathbb{N}$ such that ϕ is logically equivalent to a CNF of treewidth k.

Observation

ClauseEntailment *compiles in fpt-size wrt parameterization* mintw.

Proof.

Let ϕ be a proposition using n variables. Let ϕ' be an irredundant prime implicate normal form of ϕ with minimum treewidth (among all irredundant prime implicate normal forms of ϕ). Then, $\mathrm{tw}(\phi') = \mathrm{mintw}(\phi, \delta) = k$*. Take $c(\phi)$ to be the join tree form (a certain CNF) of a small tree decomposition of ϕ' (computable, hard). Then* $|c(\phi)| \leq k2^k \cdot n$. □

Conjecture

ClauseEntailment *not in fpt-time neither compiles in kernel-size wrt parameterization* mintw.

Fpt-Size Incompilability

Parameterization $\text{clsize}(\phi, \delta) = |\delta|$ is the number of literals in clause δ.

Fpt-Size Incompilability

Parameterization $\text{clsize}(\phi, \delta) = |\delta|$ is the number of literals in clause δ.

Observation
ClauseEntailment *does not compile in fpt-size prime implicate form wrt parameterization* clsize.

Fpt-Size Incompilability

Parameterization $\text{clsize}(\phi, \delta) = |\delta|$ is the number of literals in clause δ.

Observation
CLAUSEENTAILMENT *does not compile in fpt-size prime implicate form wrt parameterization* clsize.

Proof.
Assume f and d witness fpt-size compilation c in prime implicate form, ie, $|c(\phi)| \leq f(|\delta|)|\phi|^d$ for all ϕ and δ. For all $m, n \in \mathbb{N}$, let

$$\phi_{mn} = \left(\bigwedge_{(i,j) \in [m] \times [n]} (x_i \vee y_{ij}) \right) \wedge \left(\bigvee_{i \in [m]} \neg x_i \right).$$

Then $|\phi_{mn}| = O(mn)$. Moreover, ϕ_{mn} has $mn + (n+1)^m \geq n^m$ prime implicates ($\{y_{11}, \ldots, y_{1n}, \neg x_1\} \times \{y_{21}, \ldots, y_{2n}, \neg x_2\} \times \cdots \times \{y_{m1}, \ldots, y_{mn}, \neg x_m\}$). Therefore $|c(\phi_{mn})| \geq n^m$. Let $|\delta| = k$ and $m, n \in \mathbb{N}$ st $f(k)|\phi_{mn}|^d < n^m \leq |c(\phi_{mn})|$. $\square$

Fpt-Size Incompilability

Parameterization $\mathrm{clsize}(\phi, \delta) = |\delta|$ is the number of literals in clause δ.

Observation

CLAUSEENTAILMENT *does not compile in fpt-size prime implicate form wrt parameterization* clsize.

Proof.

Assume f *and* d *witness fpt-size compilation* c *in prime implicate form, ie,* $|c(\phi)| \leq f(|\delta|)|\phi|^d$ *for all* ϕ *and* δ*. For all* $m, n \in \mathbb{N}$*, let*

$$\phi_{mn} = \left(\bigwedge_{(i,j) \in [m] \times [n]} (x_i \vee y_{ij}) \right) \wedge \left(\bigvee_{i \in [m]} \neg x_i \right).$$

Then $|\phi_{mn}| = O(mn)$*. Moreover,* ϕ_{mn} *has* $mn + (n+1)^m \geq n^m$ *prime implicates* $(\{y_{11}, \ldots, y_{1n}, \neg x_1\} \times \{y_{21}, \ldots, y_{2n}, \neg x_2\} \times \cdots \times \{y_{m1}, \ldots, y_{mn}, \neg x_m\})$*. Therefore* $|c(\phi_{mn})| \geq n^m$*. Let* $|\delta| = k$ *and* $m, n \in \mathbb{N}$ *st* $f(k)|\phi_{mn}|^d < n^m \leq |c(\phi_{mn})|$*.* $\square$

Conjecture

CLAUSEENTAILMENT *does not compile in fpt-size wrt parameterization* clsize.

Outline

Classical Compilation

Parameterized Compilation

Research Agenda

Propositional Logic

Compilation map (Darwiche and Marquis, 2002):

1. propositional reasoning *tasks* (entailment et cetera);
2. propositional logic *formalisms* (formulas et cetera).

A certain formalism supports certain tasks in poly-time.

Propositional Logic

Compilation map (Darwiche and Marquis, 2002):

1. propositional reasoning *tasks* (entailment et cetera);
2. propositional logic *formalisms* (formulas et cetera).

A certain formalism supports certain tasks in poly-time.

Typical complexity issues within the compilation map
(under standard hypotheses in classical complexity):

1. a formalism does not support a task in poly-time;
2. a formalism does not compile into another formalism in poly-size.

Propositional Logic

Compilation map (Darwiche and Marquis, 2002):

1. propositional reasoning *tasks* (entailment et cetera);
2. propositional logic *formalisms* (formulas et cetera).

A certain formalism supports certain tasks in poly-time.

Typical complexity issues within the compilation map
(under standard hypotheses in classical complexity):

1. a formalism does not support a task in poly-time;
2. a formalism does not compile into another formalism in poly-size.

Revisit complexity issues of the compilation map within
parameterized tractability and *parameterized compilability*.

Literature

M. Cadoli, F.M. Donini, P. Liberatore, and M. Schaerf.
Preprocessing of Intractable Problems.
Information and Computation, 176(2), 89–120, 2002.

H. Chen.
Parameterized Compilability.
In *Proceedings of IJCAI'05*, 412–417, 2005.

C. Colbourn.
The Complexity of Completing Partial Latin Squares.
Discrete Applied Mathematics, 8, 25–30, 1984.

A. Darwiche and P. Marquis.
A Knowledge Compilation Map.
Journal of Artificial Intelligence Research, 17:229–264, 2002.

G. Gogic, H. Kautz, H. Papadimitriou, and B. Selman.
The Comparative Linguistics of Knowledge Representation.
In *Proceedings of IJCAI'95*, 862–869, 1995.

P. Mathieu and J.-P. Delahaye.
A Kind of Logical Compilation for Knowledge Bases.
Theoretical Computer Science, 131(1):197–218, 1994.

B. Selman and H.A. Kautz.
Knowledge Compilation and Theory Approximation.
Journal of the ACM, 43:193–224, 1996.

Gracias por su atención!