A panorama of enumeration complexity

Yann Strozecki with Florent Capelli and Arnaud Mary

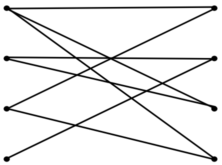
Laboratoire DAVID, Université de Versailles.

Dagstuhl Seminar on Algorithmic Enumeration

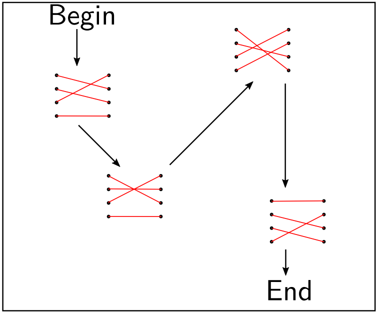
Enumeration problems

- Enumeration problems: list all solutions rather than deciding whether there is one or finding one.
- Complexity measures: total time and delay between solutions.
- Motivations: database queries, optimization, building libraries, datamining.

Perfect matching ?



Solution space:



Framework

An enumeration problem A is a function which associates to each input a set of solutions $A(x)$.

An enumeration algorithm must generate every element of $A(x)$ one after the other without repetition.

The computation model for enumeration is a RAM with uniform cost measure and an OUPTPUT instruction.

Complexity measures:

- total time
- incremental time
- delay

Complexity classes

Several classes introduced in the 80's (Johnson, Yannakakis and Papadimitriou). Allow precomputation in some classes.

1. Polynomially balanced predicate: $\textsc{EnumP}$
2. Output polynomial: $\textsc{OutputP}$
3. Incremental polynomial time: $\textsc{IncP}$
4. Polynomial delay: $\textsc{DelayP}$
5. Strong polynomial delay: $\textsc{SDelayP}$
6. Constant Delay: $\textsc{CD}$

Uniform enumeration problem

Definition

A problem A is in $\textsc{EnumP}$ if deciding whether $y \in A(x)$ is in P and if all $y \in A(x)$ are of polynomial size in x.

Equivalent to the class NP.

Uniform enumeration problem

Definition

A problem A is in $\textsc{EnumP}$ if deciding whether $y \in A(x)$ is in P and if all $y \in A(x)$ are of polynomial size in x.

Equivalent to the class NP.

Definition

A parsimonious reduction from A to B, two enumeration problems, is a pair of polynomial time computable functions f, g such that for all x, $g(x)$ is a bijection from $B(f(x))$ to $A(x)$.

- Good for hardness of enumeration of solutions NP-complete problems.
- Not general enough to prove hardness of natural problems.

Output polynomial

An output sensitive algorithm has its complexity depending on both its input and output.

Definition

A problem $A \in \textsc{EnumP}$ is in $\textsc{OutputP}$ if there is a polynomial p and a machine M which solves A in total time $O(p(|x|, |A(x)|))$.

Output polynomial

An output sensitive algorithm has its complexity depending on both its input and output.

Definition

A problem $A \in \textsc{EnumP}$ is in $\textsc{OutputP}$ if there is a polynomial p and a machine M which solves A in total time $O(p(|x|, |A(x)|))$.

$\textsc{OutputP} \neq \textsc{EnumP}$ if P $\neq$ NP, using enumeration of solutions of any NP-complete problem.

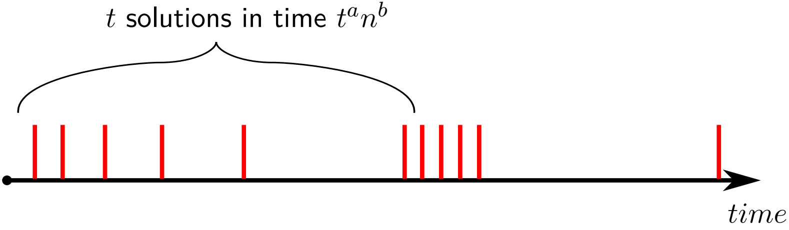
Question: is there a natural problem in $\textsc{TotalP}$ but not in the classes below?

Incremental time

A machine M enumerates A in *incremental time* $f(t)g(n)$ if on every input x, M enumerates t elements of $A(x)$ in time $f(t)g(|x|)$ for every $t \leq |A(x)|$.

Definition (Incremental polynomial time)

$\textsc{IncP}$ is the set of enumeration problems such that there is an algorithm in incremental time $O(t^a n^b)$, for inputs of size n and a, b constants.



Saturation algorithm

Most algorithms in incremental time are saturation algorithms:

- **begin** with a polynomial number of simple solutions
- **for each** k-uple of already generated solutions apply a rule to produce a new solution
- **stop** when no new solutions are found

Saturation algorithm

Most algorithms in incremental time are saturation algorithms:

- **begin** with a polynomial number of simple solutions
- **for each** k-uple of already generated solutions apply a rule to produce a new solution
- **stop** when no new solutions are found

1. Accessible vertices in a graph by flooding.
2. Generate a finite group from a set of generators.
3. Generating all the circuits of a matroid.
4. Generate all possible unions of sets:
    - $\{12, 134, 23, 14\}$
    - $\{12, 134, 1234, 23, 14\}$
    - $\{12, 134, 1234, 23, 123, 14\}$
    - $\{12, 134, 1234, 23, 123, 14, 124\}$

Relation to search problem

Search problem $\textsc{AnotherSol}{\cdot}A$
Input: x and a set of solutions $S \subset A(x)$
Output: $y \in A(x) \setminus S$ or $\sharp$ if there is none.

Relation to search problem

Search problem $\textsc{AnotherSol}{\cdot}A$
Input: x and a set of solutions $S \subset A(x)$
Output: $y \in A(x) \setminus S$ or $\sharp$ if there is none.

Theorem

An enumeration problem A is in $\textsc{IncP}$ if and only if $\textsc{AnotherSol}{\cdot}A$ can be solved in polynomial time.

Useful for hardness proof, e.g. the generation of maximal models of Horn formulas.

Relationship with total functions

Definition

A problem in TFNP is a polynomially balanced polynomial time predicate A such that for all x, $A(x)$ is not empty. An algorithm solving A must produce an element of $A(x)$ on input x.

$$\mathsf{TFNP} = \mathsf{FP}^{\mathsf{NP} \cap \mathsf{coNP}}$$

Relationship with total functions

Definition

A problem in TFNP is a polynomially balanced polynomial time predicate A such that for all x, $A(x)$ is not empty. An algorithm solving A must produce an element of $A(x)$ on input x.

$$\mathsf{TFNP} = \mathsf{FP}^{\mathsf{NP} \cap \mathsf{coNP}}$$

Proposition (Capelli, S. 2018)

TFNP = FP *if and only if* $\textsc{IncP} = \textsc{OutputP}$.

Proof: ($\Rightarrow$)Remark that $\textsc{AnotherSol}{\cdot}A$ is a TFNP problem when $A \in \textsc{OutputP}$.
($\Leftarrow$) Repeat the solutions of $A(x)$ many times to obtain an $\textsc{OutputP}$ problem.

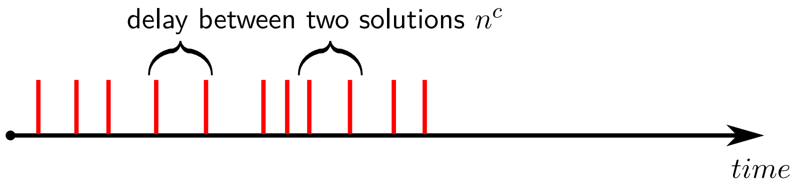
Polynomial Delay

The delay is the maximum time between the production of two consecutive solutions in an enumeration.

Definition (Polynomial delay)

DelayP is the set of enumeration problems solved by an algorithm whose delay is polynomial in the input.

$$\textsc{DelayP} \subseteq \textsc{IncP}$$



Unions in polynomial delay

Closure by union revisited.
Instance: a set $S = \{s_1, \dots s_m\}$ with $s_i \subseteq \{1, \dots, n\}$.
Problem: generate all unions of elements in S.

Unions in polynomial delay

Closure by union revisited.
Instance: a set $S = \{s_1, \ldots s_m\}$ with $s_i \subseteq \{1, \ldots, n\}$.
Problem: generate all unions of elements in S.

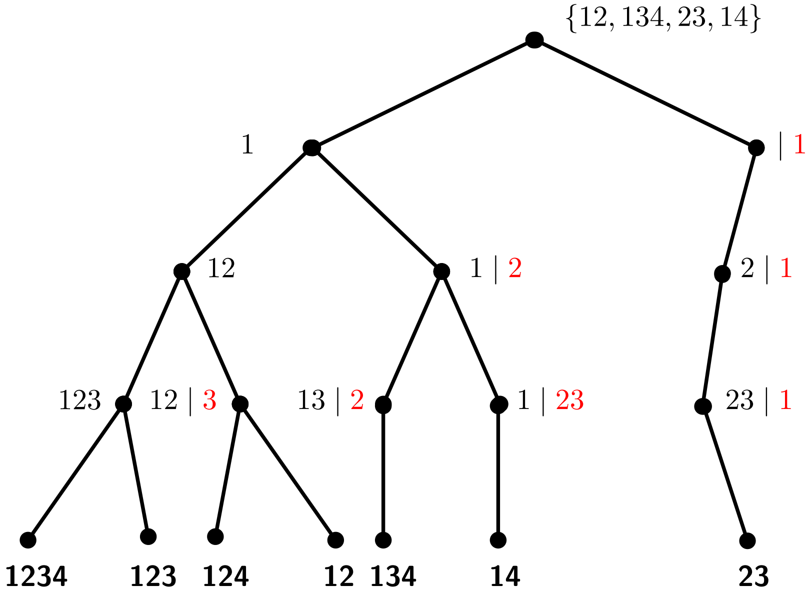
1. Recursive strategy, branch on the elements: generate the sets which contain 1, then those which do not contain 1.
2. The algorithm should not explore a branch without solutions (flashlight search), so that we can bound the delay.

Unions in polynomial delay

Closure by union revisited.
Instance: a set $S = \{s_1, \dots s_m\}$ with $s_i \subseteq \{1, \dots, n\}$.
Problem: generate all unions of elements in S.

1. Recursive strategy, branch on the elements: generate the sets which contain 1, then those which do not contain 1.
2. The algorithm should not explore a branch without solutions (flashlight search), so that we can bound the delay.
3. We must solve the extension problem: given two sets A and B is there a **solution** S **such that** $A \subseteq S$ **and** $S \cap B = \emptyset$?
4. The extension problem is easy to solve in time $O(mn)$ thus the backtrack search has delay $O(mn^2)$.

Partial solution tree



Polynomial delay methods

Flashlight search can be improved by:

- Amortizing the complexity of solving the extension problem over a branch.
- Proper choice of the variable used for branching.

The extension problem can be used to prove limited hardness results.

Polynomial delay methods

Flashlight search can be improved by:

- Amortizing the complexity of solving the extension problem over a branch.
- Proper choice of the variable used for branching.

The extension problem can be used to prove limited hardness results.

Other methods:

1. Solutions organized in a tree (models of a $2-CNF$).
2. Reverse search: solutions organized in a graph (maximal cliques).

Relationship between incremental and polynomial delay

Definition (Incremental polynomial time hierarchy)

A problem $A \in \textsc{EnumP}$ is in $\textsc{IncP}_a$ if there is a machine M which solves it in incremental time $O(t^a n^b)$ for some constant b.

Relationship between incremental and polynomial delay

Definition (Incremental polynomial time hierarchy)

A problem $A \in \textsc{EnumP}$ is in $\textsc{IncP}_a$ if there is a machine M which solves it in incremental time $O(t^a n^b)$ for some constant b.

Proposition

$\textsc{IncP}_1 = \textsc{DelayP}$.

Use amortization of solutions, exponential memory.

Are IncP_1 and DelayP really equal?

The main difference between $\textsc{IncP}_1$ and $\textsc{DelayP}$ is the regularity of the delays or equivalently the memory usage.

Are IncP_1 and DelayP really equal?

The main difference between $\textsc{IncP}_1$ and $\textsc{DelayP}$ is the regularity of the delays or equivalently the memory usage.

Theorem (Capelli, S. 2018)

Let I be an incremental linear algorithm for A using polynomial space such that for all t, in the first t solutions generated, a polynomial fraction are generated with constant delay. Then $A \in \textsc{DelayP}$ with polynomial space.

Proof: Simulate the algorithm at different points in time and use the parts with high density of solutions to compensate for sparse parts.

Are IncP_1 and DelayP really equal?

The main difference between $\textsc{IncP}_1$ and $\textsc{DelayP}$ is the regularity of the delays or equivalently the memory usage.

Theorem (Capelli, S. 2018)

Let I be an incremental linear algorithm for A using polynomial space such that for all t, in the first t solutions generated, a polynomial fraction are generated with constant delay. Then $A \in \textsc{DelayP}$ with polynomial space.

Proof: Simulate the algorithm at different points in time and use the parts with high density of solutions to compensate for sparse parts.

Open problem: Given an enumeration algorithm in polynomial delay and polynomial space, but with n repetitions of each solution turn it into a polynomial delay and polynomial space algorithm.

Separation of DelayP and IncP

Theorem (Capelli, S. 2018)

If ETH holds, then $\textsc{IncP}_a \subsetneq \textsc{IncP}_b$ for all $a < b$.

Separation of DelayP and IncP

Theorem (Capelli, S. 2018)

If ETH holds, then $\textsc{IncP}_a \subsetneq \textsc{IncP}_b$ for all $a < b$.

Proof: Problem Pad_t, input φ a CNF, with 2^{nt} trivial solutions and the models of φ duplicated 2^n times.
Since $\textsc{IncP}_a = \textsc{IncP}_b$, Pad_{b-1} gives a $O(2^{\frac{a}{b}n})$ algorithm to solve SAT.
Using the better SAT algorithm, we have $Pad_{\frac{a}{b^2}} \in \textsc{IncP}_b$. Repeat this trick to contradict ETH.

Separation of DelayP and IncP

Theorem (Capelli, S. 2018)

If ETH holds, then $\textsc{IncP}_a \subsetneq \textsc{IncP}_b$ for all $a < b$.

Proof: Problem Pad_t, input φ a CNF, with 2^{nt} trivial solutions and the models of φ duplicated 2^n times.
Since $\textsc{IncP}_a = \textsc{IncP}_b$, Pad_{b-1} gives a $O(2^{\frac{a}{b}n})$ algorithm to solve SAT.
Using the better SAT algorithm, we have $Pad_{\frac{a}{b^2}} \in \textsc{IncP}_b$. Repeat this trick to contradict ETH.

Corollary

If ETH holds, then $\textsc{DelayP} \subsetneq \textsc{IncP}$.

Open question: is there a natural problem in $\textsc{IncP}$ not in $\textsc{DelayP}$? Do you have any candidate problem?

Restricting IncP: when is saturation in polynomial delay

Question

Can we solve saturation problems in polynomial delay ?

Restricting IncP: when is saturation in polynomial delay

Question

Can we solve saturation problems in polynomial delay ?

No, since saturation problems are "equal" to $\textsc{IncP}$ and $\textsc{IncP} \neq \textsc{DelayP}$.

We need to restrict the saturation rules. Since it works for the union, we consider set operations.

Set operations

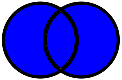
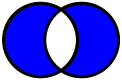
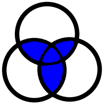
A set over $\{1, \ldots, n\}$ is represented by its characteristic vector.

A set operation is a boolean operation $\{0,1\}^k \to \{0,1\}$ applied componentwise to k boolean vectors.

Set operations

A set over $\{1, \ldots, n\}$ is represented by its characteristic vector.

A set operation is a boolean operation $\{0,1\}^k \to \{0,1\}$ applied componentwise to k boolean vectors.

$\vee$	$\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \vee \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$	$\cup$	
$+$	$\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$	$\triangle$	
$\mathrm{maj}(x, y, z)$	$\mathrm{maj}(\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$	Majority	

Closure by set operation

Let S be a set of boolean vectors of size n and $\mathcal{F}$ be a finite set of boolean operations.

Closure:

- $\mathcal{F}^0(\mathsf{S}) = \mathsf{S}$
- $\mathcal{F}^i(\mathsf{S}) = \mathcal{F}^{i-1}(\mathsf{S}) \cup$
  $\{f(v_1, \ldots, v_t) \mid v_1, \ldots, v_t \in \mathcal{F}^{i-1}(S)$ and $f \in \mathcal{F}\}$
- $Cl_{\mathcal{F}}(\mathsf{S}) = \cup_i \mathcal{F}^i(\mathsf{S})$

The enumeration problem is to list the elements of $Cl_{\mathcal{F}}(\mathsf{S})$.

Results

We classify all sets of set operations using Post lattice. It reduces the problem to a few cases:

- addition (vector space)
- disjunction, conjunction (boolean algebra)
- near unanimity terms (caracterized by projection)
- disjunction (hardest complexity-wise)

Results

We classify all sets of set operations using Post lattice. It reduces the problem to a few cases:

- addition (vector space)
- disjunction, conjunction (boolean algebra)
- near unanimity terms (caracterized by projection)
- disjunction (hardest complexity-wise)

Theorem (Mary, S. 2016)

Let $\mathcal{F}$ be a set of set operations, then listing the elements of $Cl_{\mathcal{F}}(\mathsf{S})$ can be done with delay polynomial in S.

Work either by the flashlight method, transformation to a simple SAT formula or Gray code for simple algebraic structure.

Extending the model

We want to capture more saturation operations.

1. Larger domain.
2. Non uniform operation, acting differently on each element.
3. Enumerating maximal/minimal elements only.

Extending the model

We want to capture more saturation operations.

1. Larger domain.
2. Non uniform operation, acting differently on each element.
3. Enumerating maximal/minimal elements only.

(1) For non boolean domains previous methods work but do not capture every set of operators. NP-hard extension problem, for binary associative operators we use a supergraph algorithm with exponential space.

Extending the model

We want to capture more saturation operations.

1. Larger domain.
2. Non uniform operation, acting differently on each element.
3. Enumerating maximal/minimal elements only.

(1) For non boolean domains previous methods work but do not capture every set of operators. NP-hard extension problem, for binary associative operators we use a supergraph algorithm with exponential space.

(2) If all operators act on a single coefficient (downward/upward closure), everything is polynomial delay. For operators acting on 3 coefficients, membership is NP-hard.

Hardness lurking

Generating minimal/maximal elements of a closure is not guaranteed to be in IncP. Equivalences with several fundamental problems (in Mary, S. 2018):

1. The enumeration of maximal stable sets (majority). In DelayP.
2. The enumeration of maximal independent sets in hypergraphs of dimension k (threshold functions). In IncP.
3. The enumeration of circuits of a binary matroid (symmetric difference). In IncP.

What is an efficient enumeration algorithm ?

- DelayP: equivalent to P for enumeration.
- SDelayP: polynomial delay in the size of each solution only.
- DC-LIN: constant delay, linear precomputation.

What is an efficient enumeration algorithm ?

- DelayP: equivalent to P for enumeration.
- SDelayP: polynomial delay in the size of each solution only.
- DC-LIN: constant delay, linear precomputation.
- Memory polynomial in the input.
- From one solution, the next can be output: memory proportional to the size of a solution.

What is an efficient enumeration algorithm ?

- DelayP: equivalent to P for enumeration.
- SDelayP: polynomial delay in the size of each solution only.
- DC-LIN: constant delay, linear precomputation.
- Memory polynomial in the input.
- From one solution, the next can be output: memory proportional to the size of a solution.

Relaxations:

- Randomized algorithms.
- Average (or amortized) delay.
- Approximate enumeration?

Three flavors of constant delay

The term constant delay is used to represent very different things.

- **Real** constant delay.
  Enumeration goes from a solution to the next while changing a constant number of bits. May use amortization.

Three flavors of constant delay

The term constant delay is used to represent very different things.

- **Real** constant delay.
  Enumeration goes from a solution to the next while changing a constant number of bits. May use amortization.
- Amortized constant delay.
- FPT algorithm, huge dependency in the parameter and constant size solutions.

Constant amortized time (CAT)

$$\text{Amortized} \quad \text{delay} = \frac{\text{Total Time}}{\text{Number of Solutions}}$$

Best algorithms use a constant time per object produced. Many ad-hoc methods to generate combinatorial structures. Push-out amortization [Uno].

1. Trees of size n.
2. Free trees of size n.
3. Spanning trees of a graph.
4. Matchings of a graph.

Constant delay in databases

$\Phi(\mathbf{z}, \mathbf{T})$ is a second order formula or query. Usually, the formula is fixed: data complexity.

$\textsc{Enum}{\cdot}\Phi$

Input: A σ-structure $\mathcal{S}$

Output: $\Phi(\mathcal{S}) = \{(\mathbf{z}^*, \mathbf{T}^*) : (\mathcal{S}, \mathbf{z}^*, \mathbf{T}^*) \models \Phi(\mathbf{z}, \mathbf{T})\}$

Constant delay in databases

$\Phi(\mathbf{z}, \mathbf{T})$ is a second order formula or query. Usually, the formula is fixed: data complexity.

$\textsc{Enum}{\cdot}\Phi$
Input: A σ-structure $\mathcal{S}$
Output: $\Phi(\mathcal{S}) = \{(\mathbf{z}^*, \mathbf{T}^*) : (\mathcal{S}, \mathbf{z}^*, \mathbf{T}^*) \models \Phi(\mathbf{z}, \mathbf{T})\}$

Restrictions on the logic:

- Existential FO + second order free variables.
- Acyclic conjunctive queries (lower bound using matrix multiplication).

Restrictions on the model:

- FO on low degree graphs.
- FO on bounded expansion, nowhere dense graphs.
- MSO on strings.

Constant delay in databases

$\Phi(\mathbf{z}, \mathbf{T})$ is a second order formula or query. Usually, the formula is fixed: data complexity.

$\textsc{Enum}\cdot\Phi$

Input: A σ-structure $\mathcal{S}$

Output: $\Phi(\mathcal{S}) = \{(\mathbf{z}^*, \mathbf{T}^*) : (\mathcal{S}, \mathbf{z}^*, \mathbf{T}^*) \models \Phi(\mathbf{z}, \mathbf{T})\}$

Restrictions on the logic:

- Existential FO + second order free variables.
- Acyclic conjunctive queries (lower bound using matrix multiplication).

Restrictions on the model:

- FO on low degree graphs.
- FO on bounded expansion, nowhere dense graphs.
- MSO on strings.

Open problem: prove that constant delay is impossible for FO queries over general graphs.

The case for strong polynomial delay

Strong polynomial delay is important when the input is large with regard to the size of one solution. Relevant for problems on hypergraphs.

The case for strong polynomial delay

Strong polynomial delay is important when the input is large with regard to the size of one solution. Relevant for problems on hypergraphs.

Why is it rarely considered ?

1. People are satisfied/used to polynomial delay.
2. It is harder.
3. In graph problems, typically the instance is of size $m = O(n^2)$ and the solutions are of size n: not a complexity problem.

The case for strong polynomial delay

Strong polynomial delay is important when the input is large with regard to the size of one solution. Relevant for problems on hypergraphs.

Why is it rarely considered ?

1. People are satisfied/used to polynomial delay.
2. It is harder.
3. In graph problems, typically the instance is of size $m = O(n^2)$ and the solutions are of size n: not a complexity problem.

It may be much easier to prove lower bound.

Enumerating the models of a DNF

A DNF is a disjunction of terms $\bigvee_{i=1}^{m} T_i$.
A term is a conjunction of literals over n variables.

- Enumerating the models of a term can be done in constant delay.
- Enumerating the models of a DNF in linear delay.
- m the number of terms can be much larger than n.
- Need to deal with redundancies, since a model may satisfy all terms.

Enumerating the models of a DNF

A DNF is a disjunction of terms $\bigvee_{i=1}^{m} T_i$.
A term is a conjunction of literals over n variables.

- Enumerating the models of a term can be done in constant delay.
- Enumerating the models of a DNF in linear delay.
- m the number of terms can be much larger than n.
- Need to deal with redundancies, since a model may satisfy all terms.

Weak DNF Enumeration Conjecture

Generating the models of a DNF is not in $\textsc{SDelayP}$.

Enumerating the models of a DNF

A DNF is a disjunction of terms $\bigvee_{i=1}^{m} T_i$.
A term is a conjunction of literals over n variables.

- Enumerating the models of a term can be done in constant delay.
- Enumerating the models of a DNF in linear delay.
- m the number of terms can be much larger than n.
- Need to deal with redundancies, since a model may satisfy all terms.

Strong DNF Enumeration Conjecture

There is no algorithm generating the models of a DNF in delay $o(m)$ where m is the number of terms.

Partial results on DNF

Theorem (Capelli, S. 2018)

The models of a monotone DNF can be generated in strong polynomial delay and exponential space.

Open question: Is it in strong polynomial delay and space ?

Partial results on DNF

Theorem (Capelli, S. 2018)

The models of a monotone DNF can be generated in strong polynomial delay and exponential space.

Open question: Is it in strong polynomial delay and space ?

Theorem (Capelli, S. 2018)

The models of a DNF can be generated in amortized delay $O(\sqrt{m}n)$.

Open question: Evaluate the smallest number of models of a DNF with m terms.

Partial results on DNF

Theorem (Capelli, S. 2018)

The models of a monotone DNF can be generated in strong polynomial delay and exponential space.

Open question: Is it in strong polynomial delay and space ?

Theorem (Capelli, S. 2018)

The models of a DNF can be generated in amortized delay $O(\sqrt{m}n)$.

Open question: Evaluate the smallest number of models of a DNF with m terms.

Theorem (Capelli, S. 2018)

The models of a $k-DNF$ can be generated in delay $2^{O(k)}$.

Summary

$$\mathrm{CD} \subseteq \textsc{SDelayP} \subseteq \textsc{DelayP} \subsetneq \textsc{IncP} \subsetneq \textsc{OutputP} \subsetneq \textsc{EnumP}$$

Conditional separation under complexity hypotheses as P $\neq$ NP, TFNP $\neq$ FP and ETH.

Summary

$$\mathrm{CD} \subsetneq \textsc{SDelayP} \subsetneq \textsc{DelayP} \subsetneq \textsc{IncP} \subsetneq \textsc{OutputP}$$

If we remove the condition to be in $\textsc{EnumP}$: unconditional separation.

A few interesting open problems

Prove lower bounds on classical problems using (S)ETH:

1. Minimal hitting sets of hypergraphs: delay of $m^{O(\log(m))}$. In $\textsc{IncP}$?
2. Minimal hitting sets of k-regular hypergraphs in $\textsc{IncP}_{k+2}$. Optimal?
3. Maximal cliques of a graph in delay $O(mn)$. Optimal?
4. Circuits of a binary matroids in $\textsc{IncP}_2$. Not in $\textsc{IncP}_1$?

A few interesting open problems

Prove lower bounds on classical problems using (S)ETH:

1. Minimal hitting sets of hypergraphs: delay of $m^{O(\log(m))}$. In $\textsc{IncP}$?
2. Minimal hitting sets of k-regular hypergraphs in $\textsc{IncP}_{k+2}$. Optimal?
3. Maximal cliques of a graph in delay $O(mn)$. Optimal?
4. Circuits of a binary matroids in $\textsc{IncP}_2$. Not in $\textsc{IncP}_1$?

Food for thought:

1. Settle $\textsc{IncP}_1$ vs $\textsc{DelayP}$.
2. A grammar for composing constant/polynomial delay algorithms: better algorithms and reductions.
3. Revisiting classical problems with amortized delay in mind.
4. Representative or user guided enumeration: generate a good covering of the solution space.

Thanks !

Questions ?

Link between uniform generators and enumeration

A uniform generator for the problem A is an algorithm, which given x samples the elements of $A(x)$ with uniform probability.

Theorem

If $A \in \textsc{EnumP}$ has a polytime uniform generator, then A is in randomized $\textsc{DelayP}$.

The space is proportional to the number of solutions but can be improved if we accept repetitions.

Proposition

If $A \in \textsc{EnumP}$ has a polytime uniform generator, then there is an enumeration algorithm in randomized $\textsc{IncP}_1$ with repetitions and polynomial space.