# Enumeration Complexity: Looking for Tractability

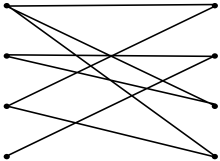
Yann Strozecki

Reading Group. Theoretical foundations of computer systems.
Simons institute

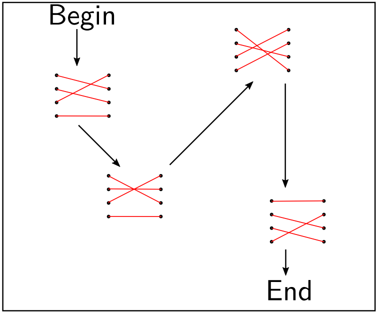
# Enumeration problems

- **Enumeration problems**: list all solutions rather than deciding whether there is one or finding one.
- **Complexity measures**: total time, delay between solutions, space.
- **Motivations**: database, logic, counting, optimization, biology, chemistry, datamining . . .

Input: a graph.



Output: perfect matchings.



# 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. Support efficient data structures.

**Complexity measures:**

- total time
- incremental time
- delay
- space

**Parameters:**

- input size
- output size
- single solution size

# Complexity classes

Several complexity classes introduced in the $80$'s [Johnson et al.] to answer the question **what is tractability** in enumeration?

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

# Polytime testing

**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 of NP for enumeration.

# Polytime testing

**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 of NP for enumeration.

**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 $A(x)$.

- Useful to prove hardness of enumerating solutions of NP-complete problems.
- Not general enough to prove hardness of natural problems.

# Tractability and EnumP

Restriction compared to the polynomial hierarchy for enumeration [Creignou et al.].

# Tractability and EnumP

Restriction compared to the polynomial hierarchy for enumeration [Creignou et al.].

Not a relevant notion of tractability:

1. No algorithm out of bruteforce.
2. Finding traces of SAT formulas or maximal H-free edge induced subgraphs are not in $\textsc{EnumP}$ but easy to solve.
3. Useful for hardness.

# 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 $O(p(|x|, |A(x)|))$.

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

# OutputP and tractability

Relevant measure of tractability because it depends on the number of solutions. Many limitations:

- All solutions must be computed (certificate of optimality, building a library).
- Should not be too many solutions and the degree of the polynomial complexity is critical.
- No hardness result and very few problems known in this class.

# OutputP and tractability

Relevant measure of tractability because it depends on the number of solutions. Many limitations:

- All solutions must be computed (certificate of optimality, building a library).
- Should not be too many solutions and the degree of the polynomial complexity is critical.
- No hardness result and very few problems known in this class.

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

# OutputP and tractability

Relevant measure of tractability because it depends on the number of solutions. Many limitations:

- All solutions must be computed (certificate of optimality, building a library).
- Should not be too many solutions and the degree of the polynomial complexity is critical.
- No hardness result and very few problems known in this class.

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

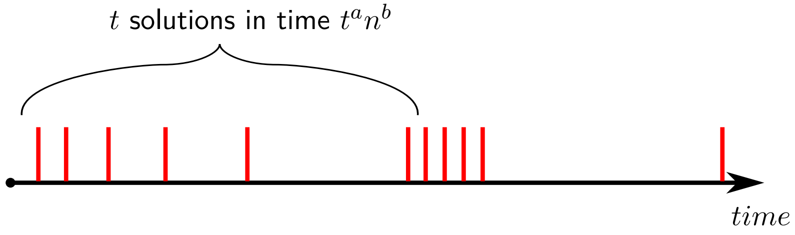
Dualization in distributive lattices [Defrain et al.].

# 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

Many incremental polynomial time algorithms are saturation algorithms:

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

# Saturation algorithm

Many incremental polynomial time algorithms are saturation algorithms:

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

1. Accessible vertices in a graph by flooding.
2. Determinization of an automata.
3. Generating all the circuits of a matroid.
4. Generate all possible unions of sets.

# Relation to a 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 a 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 $\textsc{AnotherSol}{\cdot}A \in \mathsf{FP}$.*

**Hardness proofs**: maximal models of Horn formulas [Kavvadias et al.], dualization in distributive lattice [Babin and Kuznetsov, Defrain and Nourine], repairs in databases [Kimfield et al.].

# 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. 2019)**

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

**Proof:** $(\Rightarrow)$Remark that $\textsc{AnotherSol}{\cdot}A$ is a TFNP problem when $A \in \textsc{OutputP}$.
$(\Leftarrow)$ Use many distinct copies of $A(x)$ to obtain an $\textsc{OutputP}$ problem, an $\textsc{IncP}$ algorithm allows to find one solution in FP.

# Incremental Hierarchy

**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 $b$.

# Incremental Hierarchy

**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 $b$.

**Theorem (Capelli, S. 2019)**

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

**Proof sketch:** Problem $Pad_t$, input $\varphi$ a CNF, with $2^{nt}$ trivial solutions and the models of $\varphi$ 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 $ETH$.

# Complete enumeration problem

**Corollary**

*If $ETH$ holds, then there is no problem complete for parsimonious reduction in* $\textsc{IncP}$.

**Proof:** A complete problem implies a collapse of the $\textsc{IncP}$ hierarchy to some level.

# Complete enumeration problem

**Corollary**

*If $ETH$ holds, then there is no problem complete for parsimonious reduction in* $\textsc{IncP}$.

**Proof:** A complete problem implies a collapse of the $\textsc{IncP}$ hierarchy to some level.

The result is true for most reductions (as soon as $\textsc{IncP}_a$ is stable under the reduction).

# IncP and Tractability

Relevant notion of tractability for several reasons:

- Partial enumeration: more time means more guaranteed solutions.
- Hardness results using $\textsc{AnotherSol}{\cdot}A$.
- A strict hierarchy to classify the complexity of problems inside $\textsc{IncP}$.
- The class $\textsc{IncP}_1$ as the *really tractable* problems: *linear incremental time* i.e. polynomial time per solution.

# IncP and Tractability

Relevant notion of tractability for several reasons:

- Partial enumeration: more time means more guaranteed solutions.
- Hardness results using $\textsc{AnotherSol}{\cdot}A$.
- A strict hierarchy to classify the complexity of problems inside $\textsc{IncP}$.
- The class $\textsc{IncP}_1$ as the *really tractable* problems: *linear incremental time* i.e. polynomial time per solution.

Drawbacks:

- No complete problem for the class.
- Weak regularity of the enumeration process

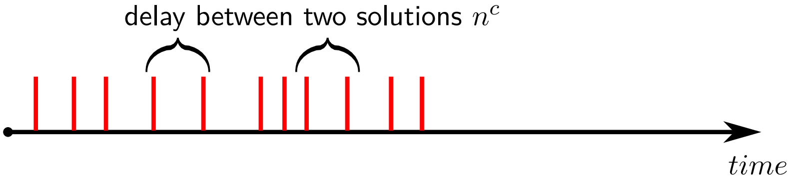
# Polynomial Delay

The delay is the maximum time between the production of two consecutive solutions given by an enumeration algorithm.

**Definition (Polynomial delay)**

A problem $A \in \textsc{EnumP}$ is in $\textsc{DelayP}$ if there is a machine $M$ which solves it on any input $x$ with delay $O(|x|^a)$.

$$\textsc{DelayP} \subseteq \textsc{IncP}_1$$



# Algorithmic Tricks for DelayP

**Proposition (Durand, S.)**

*Let $A$ and $B$ be two problems in* $\textsc{DelayP}$ *then* $A \cup B$ *is in* $\textsc{DelayP}$.

# Algorithmic Tricks for DelayP

**Proposition (Durand, S.)**

*Let $A$ and $B$ be two problems in $\textsc{DelayP}$ then $A \cup B$ is in $\textsc{DelayP}$.*

**Proof sketch:** Compute the next solution of $A$ and output it if it is not a solution of $B$ otherwise output the next solution of $B$.

If the solutions are generated in the same order, just merge them dynamically.

# Algorithmic Tricks for DelayP

**Proposition (Durand, S.)**

*Let $A$ and $B$ be two problems in* $\textsc{DelayP}$ *then* $A \cup B$ *is in* $\textsc{DelayP}$.

**Proof sketch:** Compute the next solution of $A$ and output it if it is not a solution of $B$ otherwise output the next solution of $B$.

If the solutions are generated in the same order, just merge them dynamically.

**Definition (Polynomial delay reduction)**

Reduction with only cartesian products and unions keeps $\textsc{DelayP}$ stable.

Similar to d-DNNF set circuits [Amarilli et al.].
Also in automata tools [Courcelle et al.], listing equivalence classes [Mary et al.].

# Tricks using space

Trading space for regularity.

**Proposition (Regularization with space)**

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

**Proof:** Amortize the generation of solutions, using a large queue: exponential space.

# Tricks using space

Trading space for regularity.

**Proposition (Regularization with space)**

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

**Proof:** Amortize the generation of solutions, using a large queue: exponential space.

Eliminating polynomial number of repetitions of solutions in a polynomial delay algorithm: exponential space.

**Cheater's lemma** [Carmeli et al.] and sampling to enumeration [Goldberg, Capelli and S.].

# DelayP and tractability

Most common notion of tractability in enumeration. Advantages:

- Most algorithms are naturally analyzable in term of delay.
- Regularity of enumeration?

# DelayP and tractability

Most common notion of tractability in enumeration. Advantages:

- Most algorithms are naturally analyzable in term of delay.
- Regularity of enumeration?

Drawbacks:

- No method to prove hardness.
- Should restrict space to be relevant in practice.

# Are $\mathsf{IncP}_1$ and DelayP really equal?

Let us call $\mathcal{C}^{poly}$ the class of problems in $\mathcal{C}$ which can be solved using polynomial space.

# Are $\text{IncP}_1$ and DelayP really equal?

Let us call $\mathcal{C}^{poly}$ the class of problems in $\mathcal{C}$ which can be solved using polynomial space.

Difference between $\textsc{IncP}_1$ and $\textsc{DelayP}$: regularity of enumeration or memory usage.

$$\textsc{DelayP}^{poly} = \textsc{IncP}_1^{poly}?$$

# Are IncP$_1$ and DelayP really equal?

Let us call $\mathcal{C}^{poly}$ the class of problems in $\mathcal{C}$ which can be solved using polynomial space.

Difference between $\textsc{IncP}_1$ and $\textsc{DelayP}$: regularity of enumeration or memory usage.

$$\textsc{DelayP}^{poly} = \textsc{IncP}_1^{poly}?$$

**Theorem (Capelli, S. 2019)**

*Let $A$ be a problem with a polynomial space incremental linear algorithm such that $\forall t < |A(x)|$, a polynomial fraction of the first $t$ solutions are generated with polynomial delay. Then $A \in \textsc{DelayP}^{poly}$.*

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

# Regularization without space

**Theorem (Capelli, S. (unpublished))**

*An enumerator in incremental time $p(n)t$ and space $s(n)$ can be turned into an enumerator of delay $O(p(n) * log(N))$ and space $O(s(n) * \log(N))$, where $N$ is the number of produced solutions.*

# Regularization without space

**Theorem (Capelli, S. (unpublished))**

*An enumerator in incremental time $p(n)t$ and space $s(n)$ can be turned into an enumerator of delay $O(p(n) * log(N))$ and space $O(s(n) * \log(N))$, where $N$ is the number of produced solutions.*

**Very Short Proof Sketch:**
Run $\log(N)$ copies of the enumerator. Each is in charge of the solutions in the interval of time $[2^i, 2^{i+1}]$. When a solution is found by one enumerator, it gives time to the enumerators in charge of larger intervals. Do not need to know $N$ nor $s(n)$ in advance.

# Complexity consequence

**Theorem (Capelli, S. (unpublished))**

$\textsc{DelayP}^{poly} = \textsc{IncP}_1^{poly}$

Three different takeaways:

- Incremental time is more relevant than delay.
- $\textsc{DelayP}$ is not relevant as a tractability notion: could be replaced by $\textsc{IncP}_1$.
- There is a good trick to help prove a problem is in $\textsc{DelayP}^{poly}$.

# Faster, better, tractabler

- $\textsc{DelayP}$ or $\textsc{IncP}_1$: the canonical notion of tractability for enumeration.
- $\textsc{SDelayP}$: polynomial delay in the size of the last solution.
- $\mathrm{CD}$: constant delay [Segoufin, Durand, Ruskey].

# Faster, better, tractabler

- $\textsc{DelayP}$ or $\textsc{IncP}_1$: the canonical notion of tractability for enumeration.
- $\textsc{SDelayP}$: polynomial delay in the size of the last solution.
- $\mathrm{CD}$: constant delay [Segoufin, Durand, Ruskey].
- Polynomial space.
- Polynomial time sampling.

# Faster, better, tractabler

- $\textsc{DelayP}$ or $\textsc{IncP}_1$: the canonical notion of tractability for enumeration.
- $\textsc{SDelayP}$: polynomial delay in the size of the last solution.
- $\mathrm{CD}$: constant delay [Segoufin, Durand, Ruskey].
- Polynomial space.
- Polynomial time sampling.

Help through relaxations:

- Randomized algorithms.
- Average delay: Total time / Number of solutions.
- Approximate enumeration.

# The class SDelayP

A precomputation time polynomial in the input is allowed.

**Definition (Strong polynomial delay)**

A problem $A \in \textsc{EnumP}$ is in $\textsc{SDelayP}$ if there is a machine $M$ which solves $A$ with delay $p(k)$, with $p$ a polynomial and $k$ the size of a solution.

# The class SDelayP

A precomputation time polynomial in the input is allowed.

**Definition (Strong polynomial delay)**

A problem $A \in \textsc{EnumP}$ is in $\textsc{SDelayP}$ if there is a machine $M$ which solves $A$ with delay $p(k)$, with $p$ a polynomial and $k$ the size of a solution.

**Proposition (Informal)**

*Incremental linear time $tp(k)$ is equivalent to $\textsc{SDelayP}$.*

# The class SDelayP

A precomputation time polynomial in the input is allowed.

**Definition (Strong polynomial delay)**

A problem $A \in \textsc{EnumP}$ is in $\textsc{SDelayP}$ if there is a machine $M$ which solves $A$ with delay $p(k)$, with $p$ a polynomial and $k$ the size of a solution.

**Proposition (Informal)**

*Incremental linear time $tp(k)$ is equivalent to $\textsc{SDelayP}$.*

A few examples in $\textsc{SDelayP}$:

1. $s - t$ paths in a DAG
2. $MSO$ on graphs of bounded width [Courcelle]
3. $\exists FO +$ free second order variables [Durand, S.]
4. Saturation by set operations [Mary, S.]

# Obstruction to SDelayP

One major problem to obtain a $\textsc{SDelayP}$ algorithm is dealing with non disjoint unions and repetitions in general.

# Obstruction to SDelayP

One major problem to obtain a $\textsc{SDelayP}$ algorithm is dealing with non disjoint unions and repetitions in general.

- A term is a conjunction of literals over $n$ variables.
- A DNF formula is a disjunction of $m$ terms.
- $\textsc{Enum}\cdot DNF$ is the problem of enumerating satisfying assignments of a DNF.

# Obstruction to SDelayP

One major problem to obtain a $\textsc{SDelayP}$ algorithm is dealing with non disjoint unions and repetitions in general.

- A term is a conjunction of literals over $n$ variables.
- A DNF formula is a disjunction of $m$ terms.
- $\textsc{Enum}\cdot DNF$ is the problem of enumerating satisfying assignments of a DNF.

$\textsc{Enum}\cdot DNF$ is an interesting model to study the problem of non disjoint union:

- models of terms generated in constant delay and very structured
- interesting DNF subclasses
- $\textsc{Enum}\cdot DNF$ related to knowledge representation, minimal transversal enumeration, subset membership queries, CQ + SO variables, DNF model counting . . .

# Lower Bound Conjectures for SDelayP

Delay linear in $O(mn)$ by binary partition (similar to monotone CNF [Uno]).
Can we get rid of $m$ in the complexity?

# Lower Bound Conjectures for SDelayP

Delay linear in $O(mn)$ by binary partition (similar to monotone CNF [Uno]).
Can we get rid of $m$ in the complexity?

**DNF Enumeration Conjecture**

$\textsc{Enum}\cdot DNF \notin \textsc{SDelayP}$.

**Strong DNF Enumeration Conjecture**

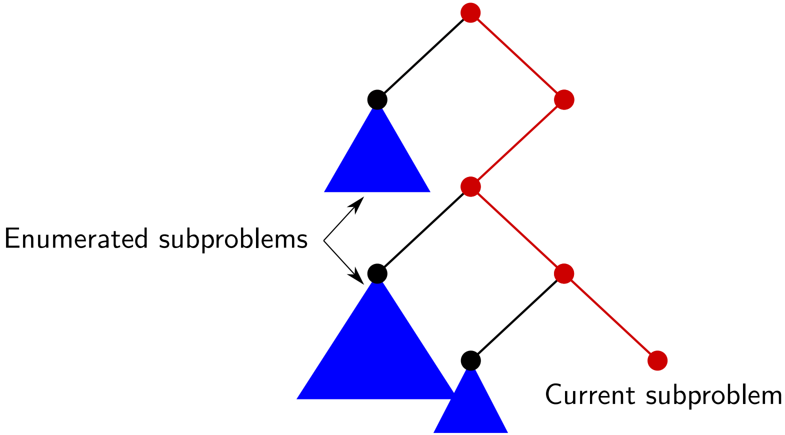
There is no algorithm generating the models of a DNF in delay $o(m)$ where $m$ is the number of terms.

# Results [Capelli, S. 2020]

| Class | Delay | Space |
|---|---|---|
| DNF | $O(\|\|D\|\|)$ | $O(\|\|D\|\|)$ |
| DNF | $O(nm^{1-\gamma})$ average delay | $O(\|\|D\|\|)$ |
| $k$-DNF | $k^{3/2}2^{2k}$ | $O(\|\|D\|\|)$ |
| Monotone DNF | $O(n^2)$, $m^2$ preprocessing | $O(sn)$ |
| Monotone DNF | $O(\log(mn))$ average delay | $O(mn)$ |

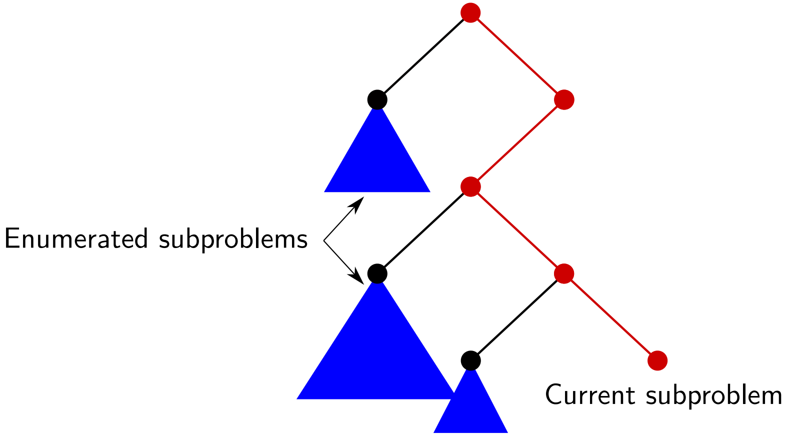
Table: Overview of the results. In this table, $D$ is a DNF, $n$ its number of variables, $m$ its number of terms and $s$ its number of models.
$\gamma = \log_3(2) > 0,63$

# Regularization of flashlight algorithms



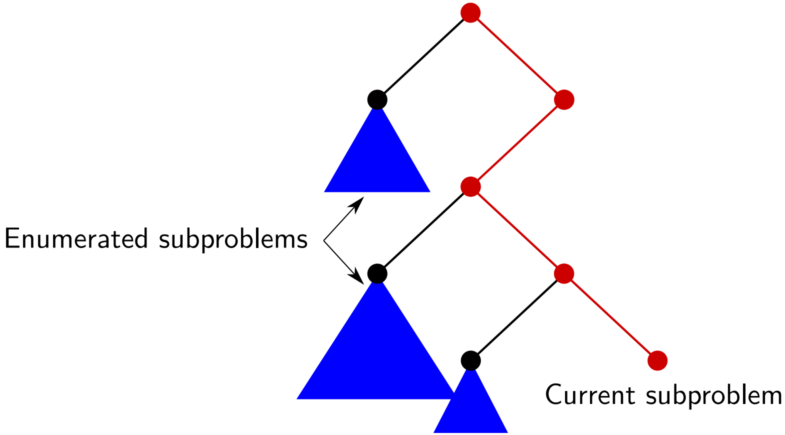

- Flashlight algorithm with average delay $a$ and delay $d$.

# Regularization of flashlight algorithms



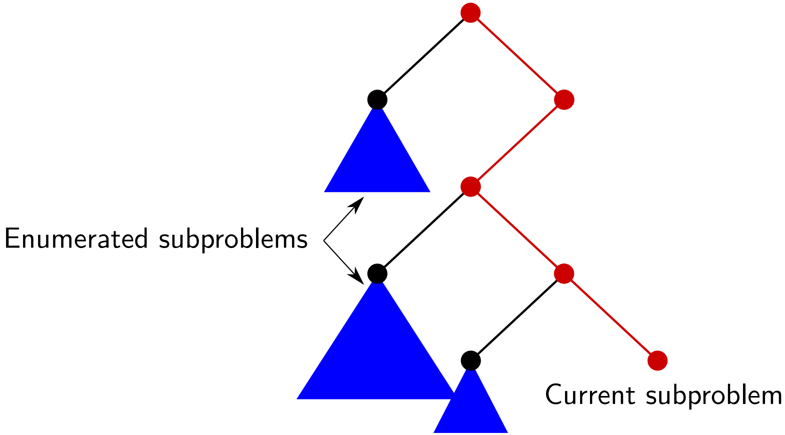

- Flashlight algorithm with average delay $a$ and delay $d$.
- Blue parts: a subproblem with $s_i$ solutions, time $as_i$.

# Regularization of flashlight algorithms



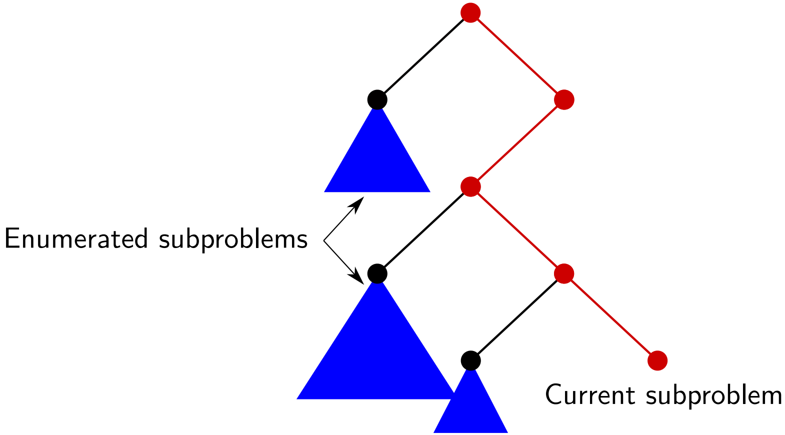

- Flashlight algorithm with average delay $a$ and delay $d$.
- Blue parts: a subproblem with $s_i$ solutions, time $as_i$.
- Red part: a path, time bounded by $d$.

# Regularization of flashlight algorithms




- Flashlight algorithm with average delay $a$ and delay $d$.
- Blue parts: a subproblem with $s_i$ solutions, time $as_i$.
- Red part: a path, time bounded by $d$.
- $\sum s_i$ solutions in $a \sum s_i + d$: $\textsc{IncP}_1$-enumerator.

# Regularization of flashlight algorithms




- Flashlight algorithm with average delay $a$ and delay $d$.
- Blue parts: a subproblem with $s_i$ solutions, time $as_i$.
- Red part: a path, time bounded by $d$.
- $\sum s_i$ solutions in $a \sum s_i + d$: $\textsc{IncP}_1$-enumerator.
- Regularized to a delay in $O(\log(N)a)$.

# Solving Enum$\cdot DNF$ with regularity

Applying the method of the previous slide to the algorithms designed for $\textsc{Enum}\cdot DNF$, we obtain the following theorems.

**Strong DNF Enumeration Conjecture is false**

There is an algorithm solving $\textsc{Enum}\cdot DNF$ in delay $O(n^2m^{1-\gamma})$ and linear space.

# Solving Enum$\cdot DNF$ with regularity

Applying the method of the previous slide to the algorithms designed for $\textsc{Enum}\cdot DNF$, we obtain the following theorems.

**Strong DNF Enumeration Conjecture is false**

There is an algorithm solving $\textsc{Enum}\cdot DNF$ in delay $O(n^2 m^{1-\gamma})$ and linear space.

**Theorem**

*There is an algorithm solving $\textsc{Enum}\cdot DNF$ for monotone formulas, in delay $\tilde{O}(n)$ and linear space.*

# SDelayP and tractability

It is a relevant notion of tractability when:

1. Large input with regard to the size of one solution: hypergraph problems, implicit input.
2. When solution size is "constant", could replace the "FPT" constant delay.
3. Doing infinite enumeration, the size of the solutions grows arbitrarily.
4. Proving lower bound of the form $A \notin \mathrm{SD\textsc{elay}P}$ should be easier.

# SDelayP and tractability

It is a relevant notion of tractability when:

1. Large input with regard to the size of one solution: hypergraph problems, implicit input.
2. When solution size is "constant", could replace the "FPT" constant delay.
3. Doing infinite enumeration, the size of the solutions grows arbitrarily.
4. Proving lower bound of the form $A \notin \mathrm{S}\textsc{Delay}\mathrm{P}$ should be easier.

Drawbacks:

1. In graph problems, the instance is typically of size $m = O(n^2)$ and the solutions are of size $n$: not a complexity problem.
2. Harder to obtain: not allowed to check the complete input between two solutions.
3. People are not familiar with this notion.

# Summary

$$\text{SDelayP} \subseteq \text{DelayP} = \text{IncP}_1 \subsetneq \text{IncP} \subsetneq \text{OutputP} \subsetneq \text{EnumP}$$

Conditional separation under complexity hypotheses: $\mathsf{P} \neq \mathsf{NP}$, $\mathsf{TFNP} \neq \mathsf{FP}$ and ETH.

# Summary

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

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

# Open problems: hardness

1. $\textsc{DelayP} \neq \textsc{SDelayP}$?
2. Existence of a complete problem in $\textsc{OutputP}$ or $\textsc{IncP}_1$?
3. Logical characterization of $\textsc{IncP}_1$, $\textsc{SDelayP}$?

# Open problems: hardness

1. $\textsc{DelayP} \neq \textsc{SDelayP}$?
2. Existence of a complete problem in $\textsc{OutputP}$ or $\textsc{IncP}_1$?
3. Logical characterization of $\textsc{IncP}_1$, $\textsc{SDelayP}$?

Lower bounds ($\textsc{SDelayP}$, $\textsc{IncP}_i$) or fine grained complexity for real problems:

1. Minimal hitting sets of hypergraphs: delay of $m^{O(\log(m))}$.
2. Minimal hitting sets of $k$-regular hypergraphs in $\textsc{IncP}_{k+2}$.
3. Maximal cliques of a graph in $\textsc{IncP}_1$.
4. Circuits of a binary matroids in $\textsc{IncP}_2$.
5. Models of a DNF in $\textsc{IncP}_1$.

Questions ???

# Four flavors of constant delay

The term constant delay is used to denote different things.

- **Real** constant delay, Gray code like algorithms.
  Enumeration goes from a solution to the next while changing a constant number of bits.

# Four flavors of constant delay

The term constant delay is used to denote different things.

- **Real** constant delay, Gray code like algorithms.
  Enumeration goes from a solution to the next while changing a constant number of bits.
- Allow dynamic amortization (generalized OUPTUT instruction).

# Four flavors of constant delay

The term constant delay is used to denote different things.

- **Real** constant delay, Gray code like algorithms.
  Enumeration goes from a solution to the next while changing a constant number of bits.
- Allow dynamic amortization (generalized OUPTUT instruction).
- Constant amortized time (CAT) algorithms. Generation of combinatorial structures of a given size, subgraphs of graphs. Pushout amortization [Uno].

# Four flavors of constant delay

The term constant delay is used to denote different things.

- **Real** constant delay, Gray code like algorithms.
  Enumeration goes from a solution to the next while changing a constant number of bits.
- Allow dynamic amortization (generalized OUPTUT instruction).
- Constant amortized time (CAT) algorithms. Generation of combinatorial structures of a given size, subgraphs of graphs. Pushout amortization [Uno].
- FPT algorithm, arbitrary dependency in the parameter. Many examples from logic/database (data complexity)in surveys by [Segoufin, Durand]. Often polynomial number of solutions: restricting preprocessing is fundamental.