Complexity of enumeration: saturation problems

Yann Strozecki

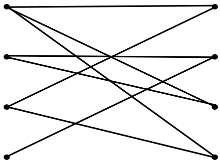
Université de Versailles St-Quentin-en-Yvelines
Laboratoire DAVID

February 2016, Séminaire MAGMAT

Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.

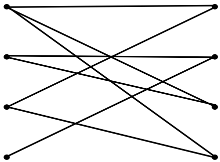
Perfect matching ?



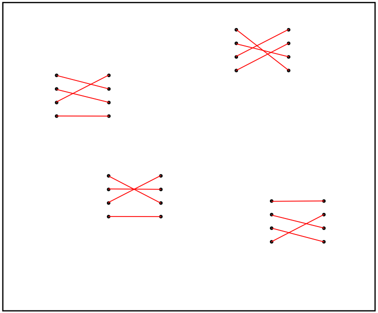
Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.

Perfect matching ?



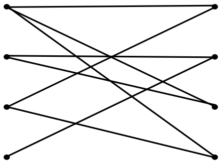
Solution space:



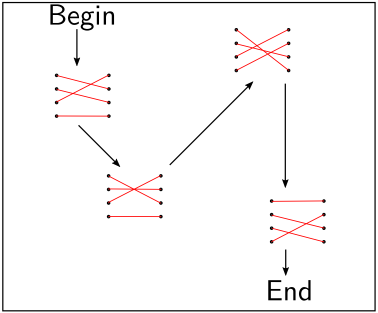
Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.

Perfect matching ?



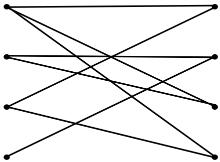
Solution space:



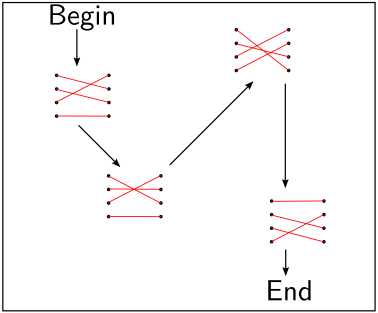
Enumeration problems

- Enumeration problems: list all solutions rather than just deciding whether there is one.
- Complexity measures: total time and delay between solutions.

Perfect matching ?



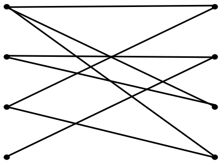
Solution space:



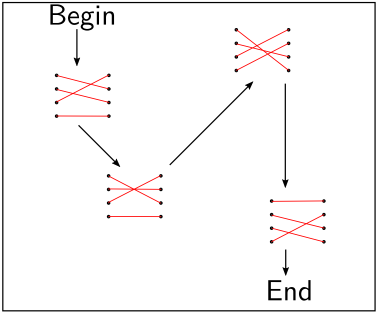
Enumeration problems

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

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.

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.

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.

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.

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.

Concrete complexity classes:

A polynomial time precomputation is allowed.

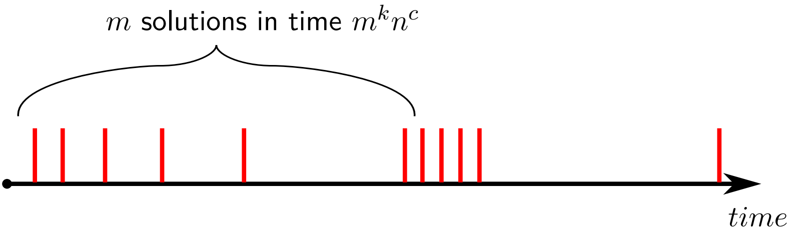
1. Polynomial total time: $\textsc{TotalP}$ (Minimal hitting set)
2. Incremental polynomial time: $\textsc{IncP}$
3. Polynomial delay: $\textsc{DelayP}$
4. Constant delay with a precomputation step: $\textsc{Constant-Delay}_{poly}$ (Database queries)

Incremental time

Definition (Incremental polynomial time)

$\mathbf{IncP_k}$ is the set of enumeration problems such that there is an algorithm which for all m produces m solutions (if they exist) from an input of size n in time $O(m^k n^c)$ with c a constant.

$$\textsc{IncP} = \bigcup_{k \geq 1} \mathbf{IncP_k}$$



Relation to research problem

Definition

The decision problem $\textsc{AnotherSol}\cdot A$ is given an instance x and a set of solutions S in $A(x)$, find a solution not in S if there is one.

Relation to research problem

Definition

The decision problem $\textsc{AnotherSol}\cdot A$ is given an instance x and a set of solutions S in $A(x)$, find a solution not in S if there is one.

Theorem

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

Relation to research problem

Definition

The decision problem $\textsc{AnotherSol}{\cdot}A$ is given an instance x and a set of solutions S in $A(x)$, find a solution not in S if there is one.

Theorem

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

The other enumeration classes cannot be related to decision problems. Hard to use classical notions such as completeness.

Saturation algorithm

Most algorithms with an incremental delay 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 with an incremental delay 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 some sets:

Saturation algorithm

Most algorithms with an incremental delay 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 some sets:
   - $\{12, 134, 23, 14\}$
   - $\{12, 134, 1234, 23, 14\}$
   - $\{12, 134, 1234, 23, 123, 14\}$
   - $\{12, 134, 1234, 23, 123, 14, 124\}$

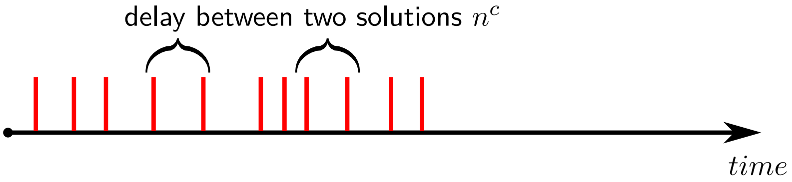
Polynomial Delay

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

Definition (Polynomial delay)

$\textsc{DelayP}$ is the set of enumeration problems such that there is an algorithm whose delay is polynomial in the input.

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



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 $S = \{s_1, \dots s_m\}$ with $s_i \subseteq \{1, \dots, n\}$.
Problem: generate all unions of elements in S.

1. Recursive strategy, enumerate first the solution which contains 1, then those which do not contain 1.

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, enumerate first the solution which contains 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.

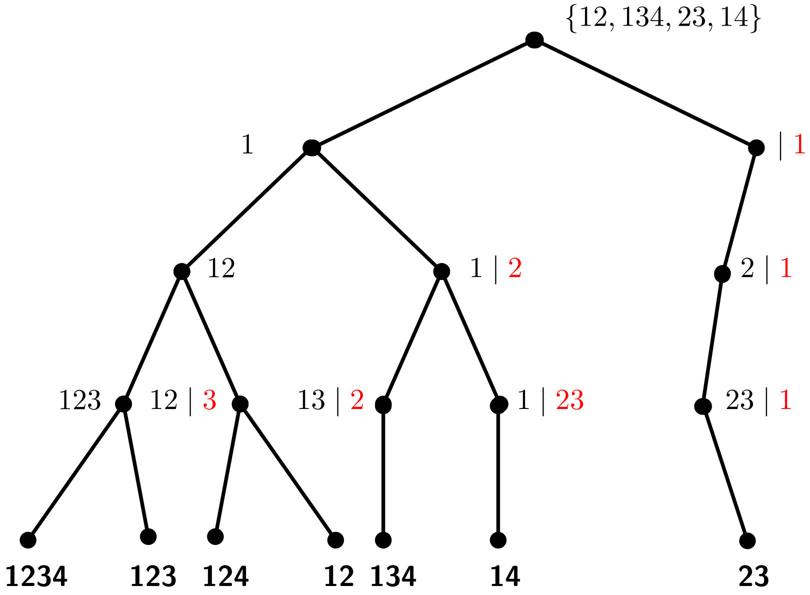
1. Recursive strategy, enumerate first the solution which contains 1, then those which do not contain 1.
2. The algorithm should not explore a branch without solutions (flashlight search), so that we can bound the delay.
3. We must solve the extension problem: given two sets A and B is there a **solution** S **such that** $A \subseteq S$ **and** $S \cap B = \emptyset$?

Unions in polynomial delay

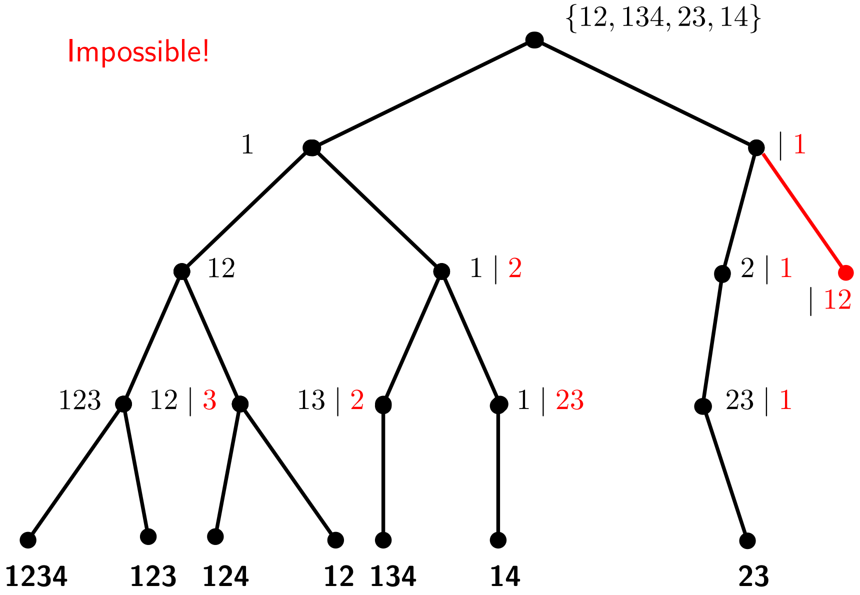
Closure by union revisited.
Instance: a set $S = \{s_1, \dots s_m\}$ with $s_i \subseteq \{1, \dots, n\}$.
Problem: generate all unions of elements in S.

1. Recursive strategy, enumerate first the solution which contains 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. This problem is easy to solve in time $O(mn)$.

Partial solution tree



Partial solution tree



Backtrack search

The backtrack search method is general. To most enumeration problem A we can associate $\textsc{Ext}{\cdot}A$ as in the previous slide.

Backtrack search

The backtrack search method is general. To most enumeration problem A we can associate $\textsc{Ext}{\cdot}A$ as in the previous slide.

Proposition

There is a polynomial delay to solve the enumeration problem A if $\textsc{Ext}{\cdot}A$ is in P.

Backtrack search

The backtrack search method is general. To most enumeration problem A we can associate $\textsc{Ext}{\cdot}A$ as in the previous slide.

Proposition

There is a polynomial delay to solve the enumeration problem A if $\textsc{Ext}{\cdot}A$ is in P.

Many applications:

- Generate all subgraphs with some constraints.
- Interpolate polynomials.
- Fold graphs.
- Generate solutions of formulas.

Can be improved by playing with the order of the variable chosen to be fixed.

Separation of complexity classes

Separation:

$$\textsc{DelayP} \subsetneq \textsc{IncP} \subsetneq \textsc{TotalP} \subsetneq \textsc{EnumP}$$

Conditional separation under complexity hypotheses.

Separation of complexity classes

Separation:

$$\text{DelayP} \subsetneq \text{IncP} \subsetneq \text{TotalP} \subsetneq \text{EnumP}$$

Conditional separation under complexity hypotheses.

1. If $\mathsf{P} = \mathsf{NP}$ everything collapses.
2. $\text{IncP} \neq \text{TotalP}$ if $\mathsf{P} \neq \mathsf{NP} \cap \mathsf{coNP}$ using problems with always a solution but an hard to find one.
3. $\text{TotalP} \neq \text{EnumP}$ if $\mathsf{P} \neq \mathsf{NP}$, using enumeration of solutions of any NP-complete problem.

Separation modulo Exponential time hypothesis

Definition

The **E**xponential **T**ime **H**ypothesis states that 3-SAT has no algorithm in time $2^{o(n)}$ where n is the number of variables.

Separation modulo Exponential time hypothesis

Definition

The **E**xponential **T**ime **H**ypothesis states that 3-SAT has no algorithm in time $2^{o(n)}$ where n is the number of variables.

Theorem (Capelli, Durand, S.)

ETH implies $\textsc{IncP}_i \subsetneq \textsc{IncP}_{i+1}$ *for all* i *and thus* $\textsc{IncP} \neq \textsc{DelayP}$.

The proof uses a two direction connection between the complexity of solving SAT and the complexity of generating all solutions of a padded version of SAT.

Separation modulo Exponential time hypothesis

Definition

The **E**xponential **T**ime **H**ypothesis states that 3-SAT has no algorithm in time $2^{o(n)}$ where n is the number of variables.

Theorem (Capelli, Durand, S.)

ETH implies $\textsc{IncP}_i \subsetneq \textsc{IncP}_{i+1}$ *for all* i *and thus* $\textsc{IncP} \neq \textsc{DelayP}$*.*

The proof uses a two direction connection between the complexity of solving SAT and the complexity of generating all solutions of a padded version of SAT.

$\textsc{DelayP} = \textsc{IncP}_1$ using an exponential size balanced binary search tree.

Open problem: is it true in polynomial space ?

From saturation to polynomial delay

Question

Can we solve saturation problems with a polynomial delay ?

From saturation to polynomial delay

Question

Can we solve saturation problems with a polynomial delay ?

In general no, since saturation problems are equals to $\textsc{IncP}$ and we have proved $\textsc{IncP} \neq \textsc{DelayP}$.

From saturation to polynomial delay

Question

Can we solve saturation problems with a polynomial delay ?

In general no, since saturation problems are equals to $\textsc{IncP}$ and we have proved $\textsc{IncP} \neq \textsc{DelayP}$.

To make the question interesting and tractable we need to restrict the saturation rules. Since it works for the union, we will consider set operations.
Our aim is to design the largest toolbox of efficient enumeration algorithms.

Set operations

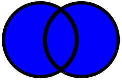
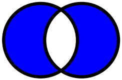
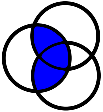
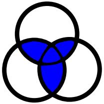
A set over $\{1, \ldots, n\}$ will be represented by its characteristic vector of size n.

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

Set operations

A set over $\{1, \ldots, n\}$ will be represented by its characteristic vector of size n.

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

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

Closure by set operation

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

Closure:

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

Closure by set operation

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

Closure:

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

Our enumeration problem is then to compute $Cl_{\mathcal{F}}(\mathcal{S})$. It can be seen as computing:

- the closure of a boolean relation by polymorphisms,
- the closure of a set system by set operations,
- the smallest hypergraph with some properties which extends the input hypergraph.

Extension problem

$\textsc{Closure}_{\mathcal{F}}$:

Input: $\mathcal{S}$ a set of vectors of size n, and a vector v of size n
Problem: decide whether $v \in Cl_{\mathcal{F}}(\mathcal{S})$.

$\textsc{Closure}_{\mathcal{F}}$ is the extension problem associated to the computation of $Cl_{\mathcal{F}}(\mathcal{S})$ (through a simple reduction).

Extension problem

$\textsc{Closure}_{\mathcal{F}}$:

Input: $\mathcal{S}$ a set of vectors of size n, and a vector v of size n
Problem: decide whether $v \in Cl_{\mathcal{F}}(\mathcal{S})$.

$\textsc{Closure}_{\mathcal{F}}$ is the extension problem associated to the computation of $Cl_{\mathcal{F}}(\mathcal{S})$ (through a simple reduction).

Goal: prove that **Closure**$_{\mathcal{F}} \in \mathsf{P}$ for as many sets $\mathcal{F}$ as possible, to use the backtrack search.

Clones and reduction

There are many finite families of boolean operations, how to reduce their number ?

Clones and reduction

There are many finite families of boolean operations, how to reduce their number ?

Definition

Let $\mathcal{F}$ be a finite set of operations, the functional clone generated by $\mathcal{F}$, denoted by $< \mathcal{F} >$, is the set of operations obtained by any composition of the operations of $\mathcal{F}$ and of the projections π_k^n defined by $\pi_k^n(x_1, \ldots, x_n) = x_k$.

For instance $(x \vee y) + x + z \ \in < \vee, + >$.

Clones and reduction

There are many finite families of boolean operations, how to reduce their number ?

Definition

Let $\mathcal{F}$ be a finite set of operations, the functional clone generated by $\mathcal{F}$, denoted by $< \mathcal{F} >$, is the set of operations obtained by any composition of the operations of $\mathcal{F}$ and of the projections π_k^n defined by $\pi_k^n(x_1, \ldots, x_n) = x_k$.

For instance $(x \vee y) + x + z \in < \vee, + >$.

Lemma

For all set of operations $\mathcal{F}$ and all set of vectors $\mathcal{S}$, $Cl_{\mathcal{F}}(\mathcal{S}) = Cl_{<\mathcal{F}>}(\mathcal{S})$.

Clones and reduction

There are many finite families of boolean operations, how to reduce their number ?

Definition

Let $\mathcal{F}$ be a finite set of operations, the functional clone generated by $\mathcal{F}$, denoted by $< \mathcal{F} >$, is the set of operations obtained by any composition of the operations of $\mathcal{F}$ and of the projections π_k^n defined by $\pi_k^n(x_1, \ldots, x_n) = x_k$.

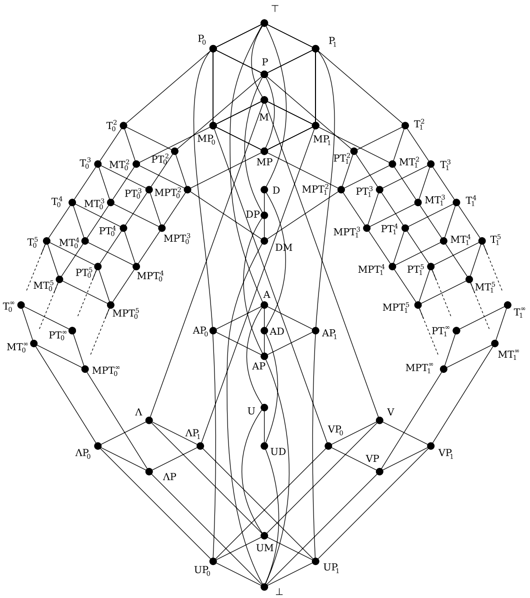
For instance $(x \vee y) + x + z \in < \vee, + >$.

Lemma

For all set of operations $\mathcal{F}$ and all set of vectors $\mathcal{S}$,
$Cl_{\mathcal{F}}(\mathcal{S}) = Cl_{<\mathcal{F}>}(\mathcal{S})$.

There are less clones than families and they are well described and organized in Post's lattice.

Post's lattice



How to reduce Post's lattice

To an operation f we can associate its dual $\overline{f}$ defined by $\overline{f}(s_1, \ldots, s_t) = \neg f(\neg s_1, \ldots, \neg s_t)$.

Proposition

The following problems can be polynomially reduced to $\textsc{Closure}_{\mathcal{F}}$*:*

1. $\textsc{Closure}_{\mathcal{F}\cup\{\mathbf{0}\}}$, $\textsc{Closure}_{\mathcal{F}\cup\{\mathbf{1}\}}$, $\textsc{Closure}_{\mathcal{F}\cup\{\mathbf{0},\mathbf{1}\}}$
2. $\textsc{Closure}_{\overline{\mathcal{F}}}$
3. $\textsc{Closure}_{\mathcal{F}\cup\{\neg\}}$ *when* $\mathcal{F} = \overline{\mathcal{F}}$

Reduced Post's lattice

Clone	Base
I_2	$\emptyset$
L_2	$x+y+z$
L_0	$+$
E_2	$\wedge$
S_{10}	$x \wedge (y \vee z)$
S_{10}^k	$Th_k^{k+1}, x \wedge (y \vee z)$
S_{12}	$x \wedge (y \rightarrow z)$
S_{12}^k	$Th_k^{k+1}, x \wedge (y \rightarrow z)$
D_2	maj
D_1	maj$, x+y+z$
M_2	$\vee, \wedge$
R_2	$x\,?\,y\;:\;z$
R_0	$\vee, +$

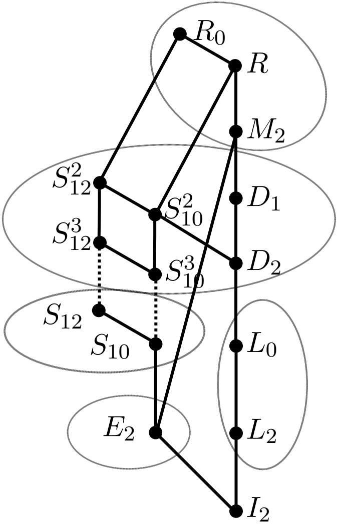


Figure: Reduced Post's lattice, the edges represent inclusions of clones

Union revisited bis

The case of $< \vee >$ is done and is equivalent to $E_2 =< \wedge >$. The delay is $O(mn^2)$, can we improve it?

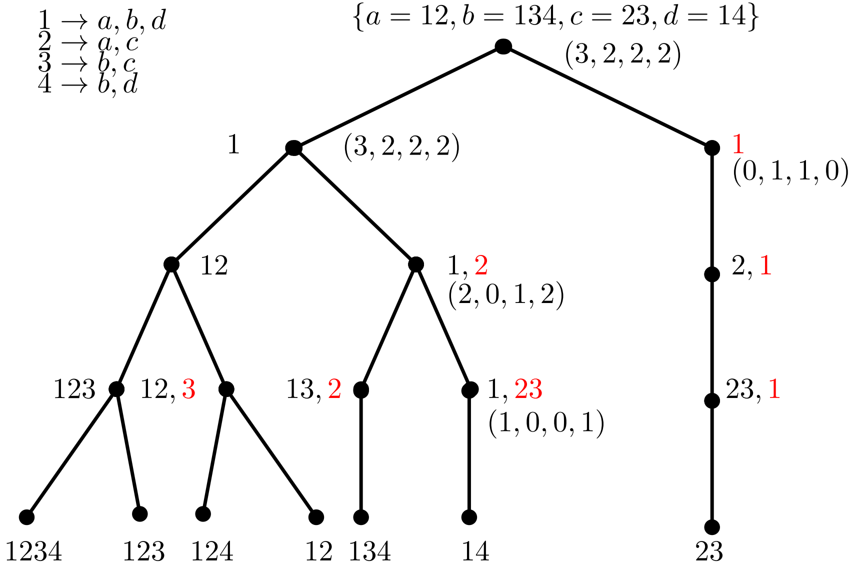
Union revisited bis

The case of $< \vee >$ is done and is equivalent to $E_2 =< \wedge >$. The delay is $O(mn^2)$, can we improve it?

- Complexity comes from solving repeatedly the extension problem.
- We can set up datastructures to solve it faster.
- During a branch of the backtrack search we go over the instance once.
- Therefore the delay is improved to $O(mn)$.

Open question: can we get rid or decrease the dependency on m?

The data structures



Symmetric difference

$L_0 =< x + y >$, $Cl_{L_0}(\mathcal{S})$ is the vector space generated by the vectors in $\mathcal{S}$.

Symmetric difference

$L_0 =< x + y >$, $Cl_{L_0}(\mathcal{S})$ is the vector space generated by the vectors in $\mathcal{S}$.

1. $\textsc{Closure}_{L_0} \in$ P since it is equivalent to solving a linear system.

Symmetric difference

$L_0 =< x + y >$, $Cl_{L_0}(\mathcal{S})$ is the vector space generated by the vectors in $\mathcal{S}$.

1. $\textsc{Closure}_{L_0} \in$ P since it is equivalent to solving a linear system.
2. We can compute a base of $Cl_{L_0}(\mathcal{S})$ in polynomial time which can be seen as a solution.

Symmetric difference

$L_0 =< x + y >$, $Cl_{L_0}(\mathcal{S})$ is the vector space generated by the vectors in $\mathcal{S}$.

1. $\textsc{Closure}_{L_0} \in$ P since it is equivalent to solving a linear system.
2. We can compute a base of $Cl_{L_0}(\mathcal{S})$ in polynomial time which can be seen as a solution.
3. This base can be turned into explicit solutions by Gray code enumeration with a delay $O(n)$.

Symmetric difference

$L_0 =< x + y >$, $Cl_{L_0}(\mathcal{S})$ is the vector space generated by the vectors in $\mathcal{S}$.

1. $\textsc{Closure}_{L_0} \in$ P since it is equivalent to solving a linear system.
2. We can compute a base of $Cl_{L_0}(\mathcal{S})$ in polynomial time which can be seen as a solution.
3. This base can be turned into explicit solutions by Gray code enumeration with a delay $O(n)$.
4. Same idea for $L_2 =< x + y + z >$, with an additionnal constraint on the elements of the basis.

All boolean functions

$M_2 =< \vee, \wedge >$, and $< M_2, \neg >$ is the set of all boolean functions.

All boolean functions

$M_2 =< \vee, \wedge >$, and $< M_2, \neg >$ is the set of all boolean functions.

- **Instance:** a set of boolean vectors $\mathcal{S} = \{s_1, \ldots, s_n\}$.

All boolean functions

$M_2 =< \vee, \wedge >$, and $< M_2, \neg >$ is the set of all boolean functions.

- **Instance:** a set of boolean vectors $\mathcal{S} = \{s_1, \ldots, s_n\}$.
- We can use the distributivity of $\vee$ and $\wedge$ to get the following normal form of a solution: $\vee_i \wedge_j l_{ij}$ with l_{ij} an element of S or its negation.

All boolean functions

$M_2 =< \vee, \wedge >$, and $< M_2, \neg >$ is the set of all boolean functions.

- **Instance:** a set of boolean vectors $\mathcal{S} = \{s_1, \ldots, s_n\}$.
- We can use the distributivity of $\vee$ and $\wedge$ to get the following normal form of a solution: $\vee_i \wedge_j l_{ij}$ with l_{ij} an element of S or its negation.
- The problem $\textsc{Closure}_{M_2,\neg}$ is in P.

All boolean functions

$M_2 =< \vee, \wedge >$, and $< M_2, \neg >$ is the set of all boolean functions.

- **Instance:** a set of boolean vectors $\mathcal{S} = \{s_1, \ldots, s_n\}$.
- We can use the distributivity of $\vee$ and $\wedge$ to get the following normal form of a solution: $\vee_i \wedge_j l_{ij}$ with l_{ij} an element of S or its negation.
- The problem $\textsc{Closure}_{M_2,\neg}$ is in P.
- We can compute the minimal $\wedge_j l_{ij}$, they are the atoms of the boolean algebra $\textsc{Closure}_{M_2,\neg}(\mathcal{S})$, which can be generated with delay $O(n)$ by Gray code.

All boolean functions

$M_2 =< \vee, \wedge >$, and $< M_2, \neg >$ is the set of all boolean functions.

- **Instance:** a set of boolean vectors $\mathcal{S} = \{s_1, \ldots, s_n\}$.
- We can use the distributivity of $\vee$ and $\wedge$ to get the following normal form of a solution: $\vee_i \wedge_j l_{ij}$ with l_{ij} an element of S or its negation.
- The problem $\textsc{Closure}_{M_2,\neg}$ is in P.
- We can compute the minimal $\wedge_j l_{ij}$, they are the atoms of the boolean algebra $\textsc{Closure}_{M_2,\neg}(\mathcal{S})$, which can be generated with delay $O(n)$ by Gray code.
- This can be done for M_2, $R_2 =< x\,?\,y : z >$ and $R_0 =< \vee, + >$.

Majority

Proposition

Let $\mathcal{S}$ be a vector set, a vector v belongs to $Cl_{<maj>}(\mathcal{S})$ if and only if for all $i, j \in [n]$, $i \neq j$, there exists $x \in \mathcal{S}$ such that $x_{i,j} = v_{i,j}$.

Idea of the proof: you build incrementally the vector v by using a sequence of vectors which have the same pairs as v.

Majority

Proposition

Let $\mathcal{S}$ be a vector set, a vector v belongs to $Cl_{<maj>}(\mathcal{S})$ if and only if for all $i, j \in [n]$, $i \neq j$, there exists $x \in \mathcal{S}$ such that $x_{i,j} = v_{i,j}$.

Idea of the proof: you build incrementally the vector v by using a sequence of vectors which have the same pairs as v.

- $\textsc{Closure}_{maj} \in \mathsf{P}$ by checking every pair of indices of the candidate vector, delay $O(mn^3)$.

Majority

Proposition

Let $\mathcal{S}$ be a vector set, a vector v belongs to $Cl_{<maj>}(\mathcal{S})$ if and only if for all $i, j \in [n]$, $i \neq j$, there exists $x \in \mathcal{S}$ such that $x_{i,j} = v_{i,j}$.

Idea of the proof: you build incrementally the vector v by using a sequence of vectors which have the same pairs as v.

- $\textsc{Closure}_{maj} \in \mathsf{P}$ by checking every pair of indices of the candidate vector, delay $O(mn^3)$.
- A linear number of pairs must be checked when a single coefficient is fixed, delay $O(mn^2)$.

Majority

Proposition

Let $\mathcal{S}$ be a vector set, a vector v belongs to $Cl_{<maj>}(\mathcal{S})$ if and only if for all $i, j \in [n]$, $i \neq j$, there exists $x \in \mathcal{S}$ such that $x_{i,j} = v_{i,j}$.

Idea of the proof: you build incrementally the vector v by using a sequence of vectors which have the same pairs as v.

- $\textsc{Closure}_{maj} \in \mathsf{P}$ by checking every pair of indices of the candidate vector, delay $O(mn^3)$.
- A linear number of pairs must be checked when a single coefficient is fixed, delay $O(mn^2)$.
- For each pair of indices, we can precompute the possible pairs of values, delay $O(n^2)$.

Thank universal algebra

Definition

An operation f is a *near unanimity* of arity k if it satisfies $f(x_1, x_2, \ldots, x_k) = x$ for each k-tuple with at most one element different from x.

Thank universal algebra

Definition

An operation f is a *near unanimity* of arity k if it satisfies $f(x_1, x_2, \ldots, x_k) = x$ for each k-tuple with at most one element different from x.

Theorem (Baker-Pixley)

Let $\mathcal{F}$ be a clone which contains a near unanimity term of arity k, then $v \in Cl_{\mathcal{F}}(\mathcal{S})$ if and only if for all set of indices I of size $k-1$, $v_I \in Cl_{\mathcal{F}}(\mathcal{S})_I = Cl_{\mathcal{F}}(\mathcal{S}_I)$.

Thank universal algebra

Definition

An operation f is a *near unanimity* of arity k if it satisfies $f(x_1, x_2, \ldots, x_k) = x$ for each k-tuple with at most one element different from x.

Theorem (Baker-Pixley)

Let $\mathcal{F}$ be a clone which contains a near unanimity term of arity k, then $v \in Cl_{\mathcal{F}}(\mathcal{S})$ if and only if for all set of indices I of size $k-1$, $v_I \in Cl_{\mathcal{F}}(\mathcal{S})_I = Cl_{\mathcal{F}}(\mathcal{S}_I)$.

The *threshold* function of arity k, denoted by Th^k_{k-1} is equal to 1 if and only if at least $k-1$ of its k arguments are equal to 1.

Corollary

For all clones $\mathcal{F}$ containing Th^k_{k-1}, $\textsc{Closure}_{\mathcal{F}} \in$ P

The result

There are two special cases $S_{10} =< x \wedge (y \vee z) >$ and $S_{12} =< x \wedge (y \to z) >$ whose proofs are similar to but not implied by the previous case.

The result

There are two special cases $S_{10} =< x \wedge (y \vee z) >$ and $S_{12} =< x \wedge (y \rightarrow z) >$ whose proofs are similar to but not implied by the previous case.

Theorem (Mary,S.)

For all sets $\mathcal{F}$ of boolean operations, $\textsc{Closure}_{\mathcal{F}} \in \mathsf{P}$.

Corollary

For all sets $\mathcal{F}$ of boolean operations, enumerating $Cl_{\mathcal{F}}$ is in $\textsc{DelayP}$.

Larger domains

The natural generalization is to consider vectors over a finite domain D with more than two elements. Every definition generalizes in a straigthforward manner.

Larger domains

The natural generalization is to consider vectors over a finite domain D with more than two elements. Every definition generalizes in a straigthforward manner.

The operations are now from D^k to D and there are many more of them. In particular the lattice of clones is uncountable and we cannot do a case by case proof.

Larger domains

The natural generalization is to consider vectors over a finite domain D with more than two elements. Every definition generalizes in a straigthforward manner.

The operations are now from D^k to D and there are many more of them. In particular the lattice of clones is uncountable and we cannot do a case by case proof.

- $D = \{0, 1, 2\}$
- $f(x, y) = x + y$ when $x + y <= 2$
- $f(x, y) = 2$
- $\textsc{Closure}_{<f>}$ is NP-hard

Tractable cases

1. Near unanimity term still implies a tractable closure problem because of Baker Pixley theorem. Generalizes to Maltsev ?

Tractable cases

1. Near unanimity term still implies a tractable closure problem because of Baker Pixley theorem. Generalizes to Maltsev ?

2. If the operation is a commutative group operation, the closure problem is in polynomial time. It can be reduced to solving several linear systems.

Tractable cases

1. Near unanimity term still implies a tractable closure problem because of Baker Pixley theorem. Generalizes to Maltsev ?

2. If the operation is a commutative group operation, the closure problem is in polynomial time. It can be reduced to solving several linear systems.

3. Associative operations yields polynomial delay algorithms by using the reverse search. It is just a depth first traversal of the solutions which can be organized as a graph of low degree. However the memory used is exponential.

Take away

Results:

- $\textsc{Closure}_{\mathcal{F}} \in$ P for all sets $\mathcal{F}$ of boolean operations.
- Enumeration of $Cl_{\mathcal{F}}$ with delay $O(n^a)$ except when $\mathcal{F} =< \vee >$.
- $\textsc{Closure}_{\mathcal{F}}$ can be NP-hard for three elements domain.

Take away

Results:

- $\textsc{Closure}_{\mathcal{F}} \in \mathsf{P}$ for all sets $\mathcal{F}$ of boolean operations.
- Enumeration of $Cl_{\mathcal{F}}$ with delay $O(n^a)$ except when $\mathcal{F} =< \vee >$.
- $\textsc{Closure}_{\mathcal{F}}$ can be NP-hard for three elements domain.

Open questions:

- Characterize the complexity of $\textsc{Closure}_{\mathcal{F}}$ for larger domains (dichotomy theorem?).
- Enumerate $Cl_{\mathcal{F}}$ when $\mathcal{F}$ is a single non commutative group operation.
- Improve the delay of enumerating $Cl_{<\vee>}$.

Thanks !

Questions ?