Enumerating models of a DNF: sublinear algorithms

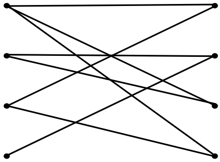
Florent Capelli and **Yann Strozecki**

Séminaire LIMOS

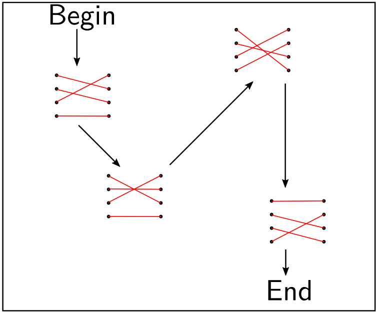
Enumeration problems

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

Perfect matchings?



Solution space:



A concrete example: enumerating integers

Problem: generate all integers in $[0, 2^n - 1]$ or all subsets of $[n]$.

A concrete example: enumerating integers

Problem: generate all integers in $[0, 2^n - 1]$ or all subsets of $[n]$.

Solution: Add one repeatedly: 0000, 0001, 0010, $\dots$

Complexity: The delay, going from a number to the next, is in $O(n)$ in the worst case:

$$01111 + 1 = 10000$$

A concrete example: enumerating integers

Problem: generate all integers in $[0, 2^n - 1]$ or all subsets of $[n]$.

Solution: Add one repeatedly: 0000, 0001, 0010, $\ldots$

Complexity: The delay, going from a number to the next, is in $O(n)$ in the worst case:

$$01111 + 1 = 10000$$

However, the average time per integer generated is constant:

$$\sum i2^{-i} \leq 2$$

Faster enumeration using Gray code

A Gray Code is an ordering of the integers in $[0, 2^n - 1]$ such that two consecutive elements differ by exactly one bit.

Faster enumeration using Gray code

A Gray Code is an ordering of the integers in $[0, 2^n - 1]$ such that two consecutive elements differ by exactly one bit.

Reflected Gray Code: recursive construction.
$\emptyset$

Faster enumeration using Gray code

A Gray Code is an ordering of the integers in $[0, 2^n - 1]$ such that two consecutive elements differ by exactly one bit.

Reflected Gray Code: recursive construction.

- 0
- 1

Faster enumeration using Gray code

A Gray Code is an ordering of the integers in $[0, 2^n - 1]$ such that two consecutive elements differ by exactly one bit.

Reflected Gray Code: recursive construction.

- 0 0
- 0 1
- 1 1
- 1 0

Faster enumeration using Gray code

A Gray Code is an ordering of the integers in $[0, 2^n - 1]$ such that two consecutive elements differ by exactly one bit.

Reflected Gray Code: recursive construction.

- 0 0
- 0 1
- 1 1
- 1 0

There is a constant time algorithm to find the next element in the Gray code:

- the number of one is even: flip the last bit
- the number of one is odd: flip the bit to the left of the rightmost 1

Framework

An enumeration problem A is a function which associates to each input x 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.

Framework

An enumeration problem A is a function which associates to each input x 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
- delay
- space

Parameters:

- input size
- output size
- single solution size

Generating unions

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

Generating unions

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

Use a saturation algorithm:

- **begin** with a polynomial number of simple solutions: the sets s_i
- **for each** k-uple of already generated solutions apply a rule to produce a new solution: produce $s \cup s'$ for each pair (s, s') of solutions
- **stop** when no new solutions are found

Unions in polynomial delay

A better algorithm to compute the closure by union.

1. Recursive strategy, branch on elements: partition solutions into those containing element 1 or not.
   Bruteforce, backtrack search, binary partition.

Unions in polynomial delay

A better algorithm to compute the closure by union.

1. Recursive strategy, branch on elements: partition solutions into those containing element 1 or not.
   Bruteforce, backtrack search, binary partition.

2. The algorithm should not explore a branch without solutions.
   Similar to branch and bound, flashlight search.

Unions in polynomial delay

A better algorithm to compute the closure by union.

1. Recursive strategy, branch on elements: partition solutions into those containing element 1 or not.
   Bruteforce, backtrack search, binary partition.
2. The algorithm should not explore a branch without solutions.
   Similar to branch and bound, flashlight search.
3. Solve the extension problem: given $A, B \subseteq \{1, \ldots, n\}$ is there **solution** A' **such that** $A \subseteq A'$ **and** $A' \cap B = \emptyset$?

Unions in polynomial delay

A better algorithm to compute the closure by union.

1. Recursive strategy, branch on elements: partition solutions into those containing element 1 or not.
   Bruteforce, backtrack search, binary partition.

2. The algorithm should not explore a branch without solutions. Similar to branch and bound, flashlight search.

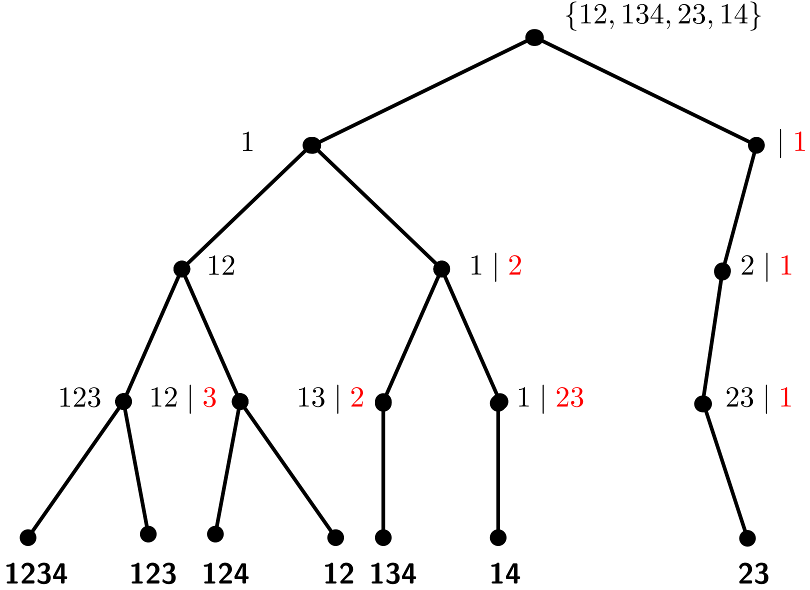
3. Solve the extension problem: given $A, B \subseteq \{1, \ldots, n\}$ is there **solution** A' **such that** $A \subseteq A'$ **and** $A' \cap B = \emptyset$?

4. Easy to solve in $O(mn)$: compute

$$A' = \bigcup_{s \in S, s \cap B = \emptyset} s$$

   Delay equal *depth of the recursive tree* time *cost of solving extension*: $O(mn^2)$.

Partial solution tree



Polynomial delay methods

Flashlight search can be improved by:

- Reducing the complexity of the extension problem or the depth of the tree
- Proper choice of the element used for branching
- Amortizing the complexity of solving the extension problem over a branch
- Amortizing the complexity over all leaves

Polynomial delay methods

Flashlight search can be improved by:

- Reducing the complexity of the extension problem or the depth of the tree
- Proper choice of the element used for branching
- Amortizing the complexity of solving the extension problem over a branch
- Amortizing the complexity over all leaves

Other methods:

1. Solutions organized in a tree (models of a $2 - CNF$).
2. Solutions organized in a connected graph, with small or enumerable neighboroods. Reverse search (maximal cliques).

What is a really efficient enumeration algorithm ?

- $\textsc{DelayP}$: equivalent to P for enumeration
- $\textsc{SDelayP}$: polynomial delay in the size of a solution + preprocessing
- CD: constant delay + preprocessing

What is a really efficient enumeration algorithm ?

- $\textsc{DelayP}$: equivalent to P for enumeration
- $\textsc{SDelayP}$: polynomial delay in the size of a solution + preprocessing
- CD: constant delay + preprocessing
- Additional condition: space polynomial in the input or a single solution
- Polynomial time sampling

What is a really efficient enumeration algorithm ?

- $\textsc{DelayP}$: equivalent to P for enumeration
- $\textsc{SDelayP}$: polynomial delay in the size of a solution + preprocessing
- CD: constant delay + preprocessing
- Additional condition: space polynomial in the input or a single solution
- Polynomial time sampling

A typical example: listing all paths in a DAG by a DFS.

Enumerating the models of a DNF

- 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 ($=$ models) of a DNF.

Why is this problem interesting?

Enumerating the models of a DNF

- 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 (= models) of a DNF.

Why is this problem interesting?

- Extremely simple: solution of terms in constant delay. Union of regular sets of solutions while dealing with repetitions.
- DNF enumeration is connected to knowledge representation, minimal transversal enumeration, subset membership queries, CQ + SO variables, DNF model counting, PAC-learning . . .

Representation of a DNF

A DNF formula D is characterized by the following parameters:

1. n the number of variables
2. m the number of terms
3. $||D||$ the sum of the sizes of the terms, $||D|| \leq nm$
4. s the number of its models

Representation of a DNF

A DNF formula D is characterized by the following parameters:

1. n the number of variables
2. m the number of terms
3. $||D||$ the sum of the sizes of the terms, $||D|| \leq nm$
4. s the number of its models

Data structure for DNF: the Trie

A trie or prefix tree is a tree labeled by letters. It represents the set of words associated to its paths from the root.

A DNF formula or a set of models can be represented by a trie. It supports insertion, lookup and deletion in time $O(n)$.

Terms and union of terms

Theorem

The models of a term can be generated with constant delay.

Use Gray code and an array to store the indices of the free variables.

Terms and union of terms

Theorem

The models of a term can be generated with constant delay.

Use Gray code and an array to store the indices of the free variables.

Simplest algorithm: for each term, generate all its models and deal with repetitions using a set data structure.

Terms and union of terms

Theorem

The models of a term can be generated with constant delay.

Use Gray code and an array to store the indices of the free variables.

Simplest algorithm: for each term, generate all its models and deal with repetitions using a set data structure.

The cost for lookup and insertion depends on the data structure:

- Array (sorted or not): $O(ns)$
- Binary search tree: $O(n \log(s))$
- Hash table: expected $O(n)$
- Trie: $O(n)$

Flashlight for DNF

Application of the flashlight method: depth of the tree n, extension problem in $O(mn)$: delay in $O(mn^2)$.

Flashlight for DNF

Application of the flashlight method: depth of the tree n, extension problem in $O(mn)$: delay in $O(mn^2)$.

Better data structure (similar to the one for monotone CNF [Uno]):

- A counter for the number of satisfiable terms in D: c_D
- For each term T a counter of falsified variables: c_T
- For each literal, the list of terms where it appears

In a branch, each literal of a term is visited once at most: delay $O(||D||)$.

Lower Bound Conjectures

We have a linear delay algorithm. Can we get rid of m in the delay?

Lower Bound Conjectures

We have a linear delay algorithm. Can we get rid of m in the delay?

DNF Enumeration Conjecture

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

Strong DNF Enumeration Conjecture

There is no algorithm solving $\textsc{Enum}{\cdot}DNF$ in delay $o(m)p(n)$ where m is the number of terms, n the number of variables and p a polynomial.

Lower Bound Conjectures

We have a linear delay algorithm. Can we get rid of m in the delay?

DNF Enumeration Conjecture

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

Strong DNF Enumeration Conjecture

There is no algorithm solving $\textsc{Enum}\cdot DNF$ in delay $o(m)p(n)$ where m is the number of terms, n the number of variables and p a polynomial.

Stronger conjectures can be made by restricting to subclasses of DNF and to average delay. We refute some of them in this presentation.

DNF with small terms

Definition

A term T is a k-term if $|T| \leq k$.
A DNF is a k-DNF if all its terms are k-terms.

DNF with small terms

Definition

A term T is a k-term if $|T| \leq k$.
A DNF is a k-DNF if all its terms are k-terms.

Solutions aplenty: A k-term has 2^{n-k} models.

A compact representation of subproblems: if D is represented by the trie $T(D)$, one can compute $T(D[x \to 0])$ and $T(D[x \to 1])$ in time $O(||D||)$.

All removed terms are stored to be able to insert them back later.

Branching along a term

Let $T = x_1 \wedge x_2 \cdots \wedge x_k$ be a term of D. The models of D can be partitionned into $k+1$ disjoint subsets, models of:

- $D \wedge \bar{x}_1$
- $D \wedge x_1 \wedge \bar{x}_2$
- $\ldots$
- $D \wedge x_1 \wedge x_2 \wedge \cdots \wedge \bar{x}_k$
- $D \wedge x_1 \wedge x_2 \cdots \wedge x_k$

Branching along a term

Let $T = x_1 \wedge x_2 \cdots \wedge x_k$ be a term of D. The models of D can be partitionned into $k+1$ disjoint subsets, models of:

- $D \wedge \bar{x}_1$
- $D \wedge x_1 \wedge \bar{x}_2$
- ...
- $D \wedge x_1 \wedge x_2 \wedge \cdots \wedge \bar{x}_k$
- $D \wedge x_1 \wedge x_2 \cdots \wedge x_k$

The models of $D \wedge x_1 \wedge x_2 \cdots \wedge x_k$ are the models of $x_1 \wedge x_2 \cdots \wedge x_k$ an can be enumerated in constant delay.

Enumeration for k-DNF

Theorem

The models of a k-DNF with n variables can be enumerated with precomputation in $O(n)$ and $O(k^{3/2}2^{2k})$ delay.

Sketch of the algorithm:

- Flashligth algorithm, *branch along a term* of the current formula.
- When a variable is fixed, *compute the trie of the reduced formula*.
- *Interleave* enumeration of models of a term T and branching along T.
- Balance the cost of maintaining a trie and the number of solutions given by a model, worst case $n = 2k$.

The number of models of a DNF

For k-DNF, having so many solutions helps.
Similar property for the general case?

The number of models of a DNF

For k-DNF, having so many solutions helps.
Similar property for the general case?

Lemma

Let $\gamma = \log_3(2)$. A DNF formula with m non empty distinct terms has at least m^γ models.

Proof sketch:
Induction on the number of variables. Cut the formula in three parts: terms with x_1, $\bar{x}_1$, whithout x_1 or $\bar{x}_1$. Evaluate how the three subformulas contribute models to the original formula and apply inequalities.

Average delay of the flashlight search

Same idea as for k-DNF, amortize the cost of branching.

Theorem

The models of a DNF can be enumerated with average delay $O(n^2 m^{1-\gamma})$ and polynomial space.

Average delay of the flashlight search

Same idea as for k-DNF, amortize the cost of branching.

Theorem

The models of a DNF can be enumerated with average delay $O(n^2 m^{1-\gamma})$ and polynomial space.

Proof sketch:

Flashlight search, reduction of the trie when fixing a variable.
Branching cost $O(mn)$, but can be amortized over m^γ models.

Improving the average delay

How to reduce the depth of the tree of partial solutions?

Improving the average delay

How to reduce the depth of the tree of partial solutions?

Theorem

The models of a DNF can be enumerated with average delay $O(nm^{1-\gamma})$ and polynomial space.

Proof sketch:

Differentiate between **fast branching** (the size of one subformula decreases by a factor $1/2$) and **slow branching**. A different way of counting the complexity of branching and amortizing it is used in the two cases.

Improving the average delay

How to reduce the depth of the tree of partial solutions?

Theorem

The models of a DNF can be enumerated with average delay $O(nm^{1-\gamma})$ and polynomial space.

Proof sketch:
Differentiate between **fast branching** (the size of one subformula decreases by a factor $1/2$) and **slow branching**. A different way of counting the complexity of branching and amortizing it is used in the two cases.

Above algorithm and the one for k-DNF can be combined.

Theorem

There is an algorithm with average delay $O(2^{3k/2})$ to enumerate the models of a k-DNF.

Monotone DNF

Problems of generating ideals in a boolean lattice given by an antichain.

- Remove redundant terms (terms included in other) in time $O(m^2)$.
- Each term has one proper model (the minimal one).
- For each term, enumerate its models in lexicographic order and store them in a set.
- When a redundant solution is found do not explore further.
- At most $O(n)$ consecutive redundant solutions before seeing a fresh one.

Monotone DNF

Problems of generating ideals in a boolean lattice given by an antichain.

- Remove redundant terms (terms included in other) in time $O(m^2)$.
- Each term has one proper model (the minimal one).
- For each term, enumerate its models in lexicographic order and store them in a set.
- When a redundant solution is found do not explore further.
- At most $O(n)$ consecutive redundant solutions before seeing a fresh one.

Open question: Can the preprocessing be improved? Can the delay be improved? Can the space be made polynomial?

Average delay for Monotone DNF

The algorithm for general DNF is adapted using two ideas.

- Each term has a minimal model not shared with the other terms. For an instance with m terms, at least m models.
- Terms of small size have many solutions: a formula with a small term cost "nothing" in the flashlight. If all terms are large represent them by their complementary.

Average delay for Monotone DNF

The algorithm for general DNF is adapted using two ideas.

- Each term has a minimal model not shared with the other terms. For an instance with m terms, at least m models.
- Terms of small size have many solutions: a formula with a small term cost "nothing" in the flashlight. If all terms are large represent them by their complementary.

Theorem

There is an algorithm to enumerate the models of a monotone DNF with polynomial space and average delay $O(mn)$.

Results [Capelli, S. 2019]

Class	**Delay**	**Space**
DNF	$O(\|\|D\|\|)$	$O(\|\|D\|\|)$
$(\star)$ DNF	$O(nm^{1-\gamma})$ average delay	$O(\|\|D\|\|)$
$(\star)$ k-DNF	$k^{3/2}2^{2k}$	$O(\|\|D\|\|)$
$(\star)$ Monotone DNF	$O(n^2)$, m^2 preprocessing	$O(sn)$
$(\star)$ 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 and m its number of terms. New contributions are annotated with $(\star)$.

Thanks!
Questions?