Unranking of Reduced Ordered Binary Decision Diagrams

arXiv:2211.04938

Antoine Genitrini[†]

with Julien Clément[*]

Workshop Computational Logic and Applications

January 13, 2023

[†]LIP6 @ Sorbonne University

[*]GREYC @ Normandie University

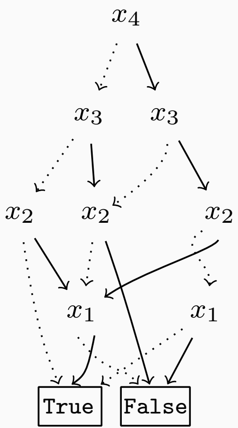
Binary Decision Diagram

Let f be a Boolean function in k variables.

A *Binary Decision Diagram* is a compact representation of f allowing to evaluate it efficiently.

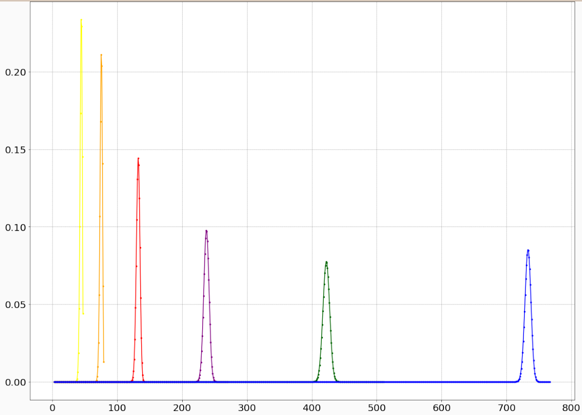
It is based on some divide-and-conquer principle.

[Wegener00]: Branching Programs and Binary Decision Diagrams
[Knuth11]: The Art of Computer Programming (vol.4)



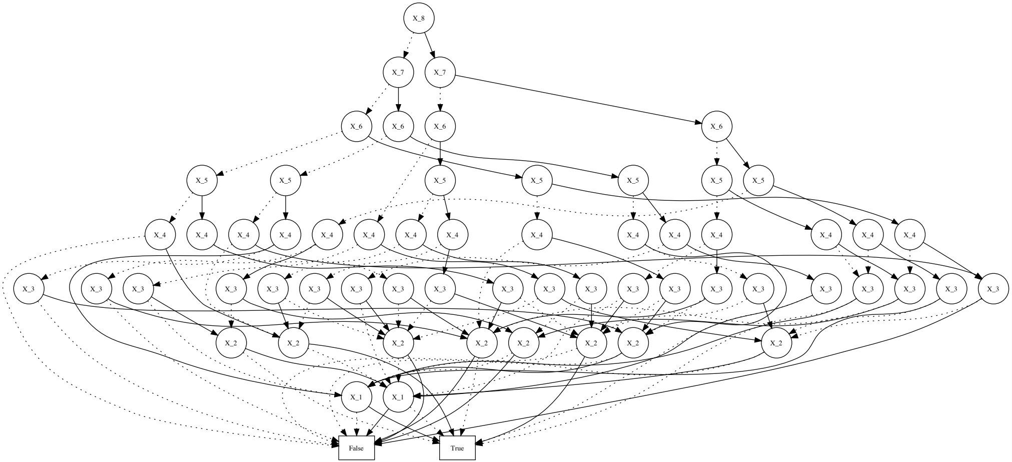
We are interested in Reduced Ordered Binary Decision Diagrams, denoted ROBDDs.
We take a point of view of a combinatorialist.

Shannon effect: ROBDD's size distribution

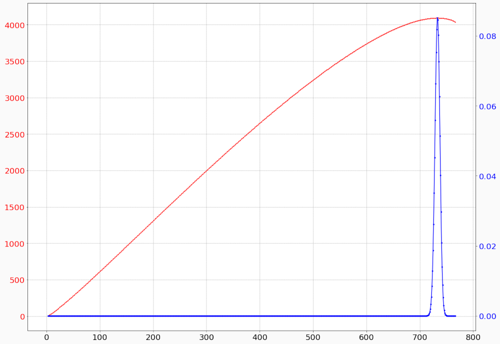


# variables	7	8	9	10	11	12
# functions	$2^{2^7} = 2^{128}$	2^{256}	2^{512}	2^{1024}	2^{2048}	2^{4096}
	$\approx 10^{38}$	$\approx 10^{77}$	$\approx 10^{154}$	$\approx 10^{308}$	$\approx 10^{616}$	$\approx 10^{1233}$

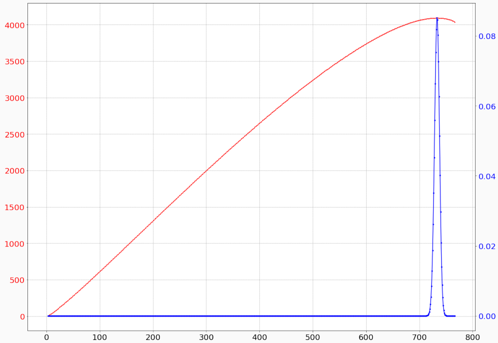
Motivations: Building ROBDDs of small size

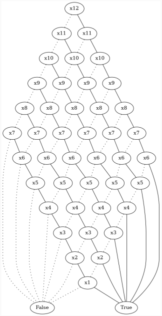


Motivations: Building ROBDDs of small sizes



Motivations: Building ROBDDs of small sizes





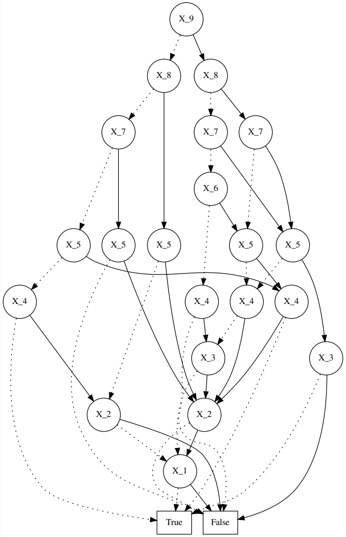
Specific functions with small ROBDDs (in k variables):

- addition in k bits,
- read-once fcts,
- symmetric fcts: $O(k^2)$.

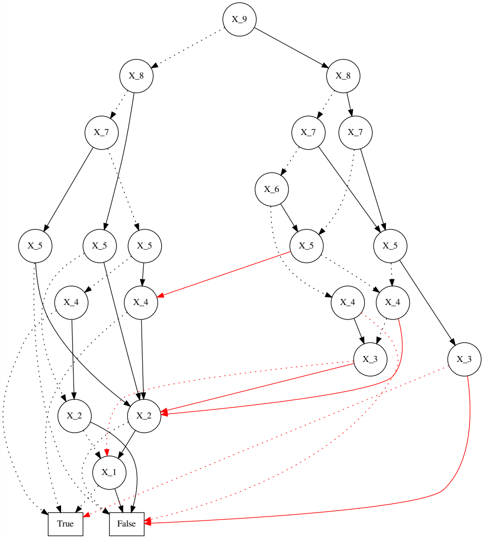
ℓ threshold functions: $7 \leq k \leq 12$; $1 \leq \ell \leq k$

9	14	17	**18**	17	14	9					
10	16	20	**22**	22	20	16	10				
11	18	23	26	**27**	26	23	18	11			
12	20	26	30	**32**	32	30	26	20	12		
13	22	29	34	37	**38**	37	34	29	22	13	
14	24	32	38	42	**44**	44	42	38	32	24	14

Why does the enumeration be difficult?

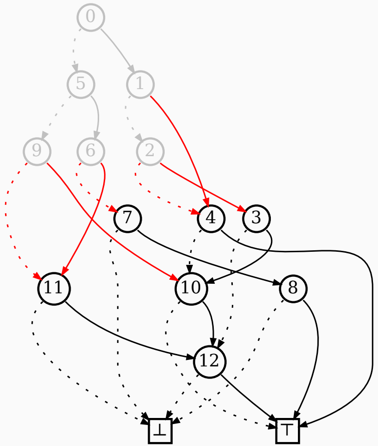


Why does the enumeration be difficult?



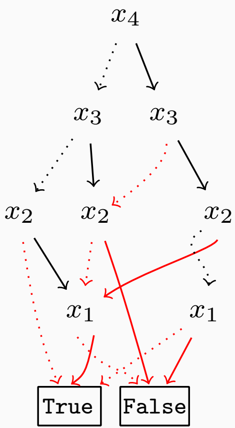
Outline of the talk

- Combinatorial preliminaries



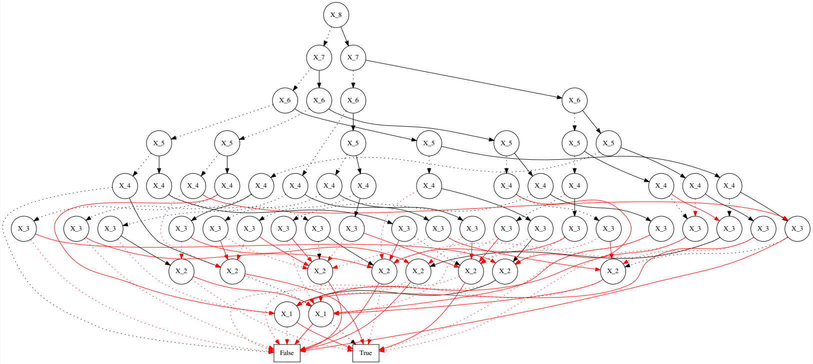
Outline of the talk

- Combinatorial preliminaries
- Iterative counting



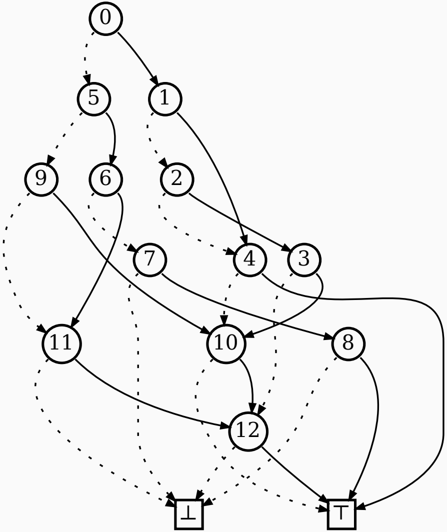
Outline of the talk

- Combinatorial preliminaries
- Iterative counting
- Unranking a ROBDD



Combinatorial preliminaries

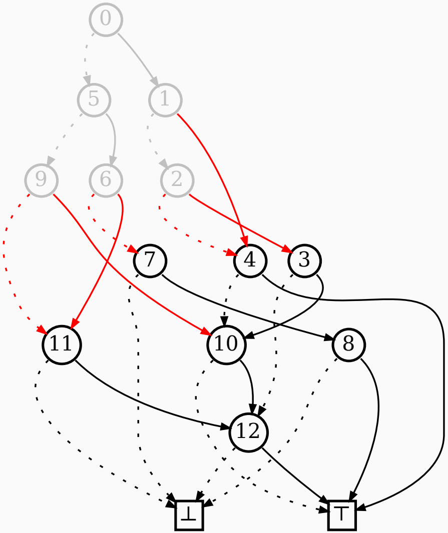
Focus on DAGs ≈ ROBDDs



Concepts:

- low/high child
- size
- layers
- constraints:
  - useful node property
  - descendants unicity in a given layer
  - acyclicity
  - accessibility
- profile $[1, 2, 3, 3, 3, 1, 2]$
- reverse postorder traversal

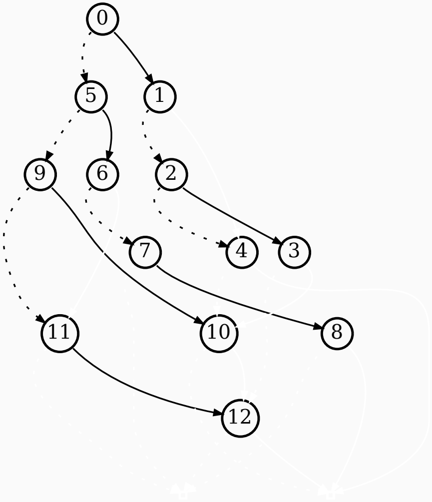
Multientry ROBDD



Concepts:

- removing upper layers multientry ROBDD
    - keeping destination of red edges
      as a multiset $\{3, 4, 4, 7, 10, 11, 11\}$

Spine of a ROBDD



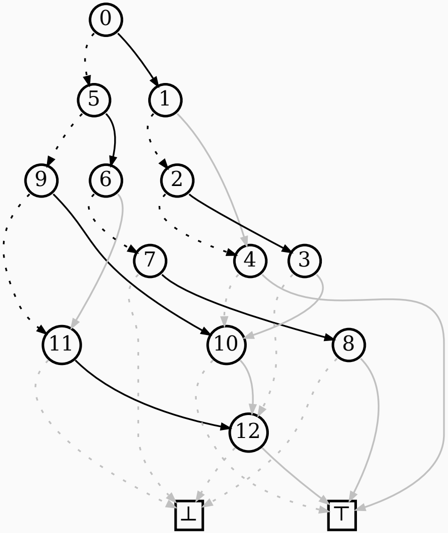
Concepts:

- the spine of a ROBDD

  spanning tree given by a depth-first search

  - ⇒ tree edges

Spine of a ROBDD



Concepts:

- the spine of a ROBDD

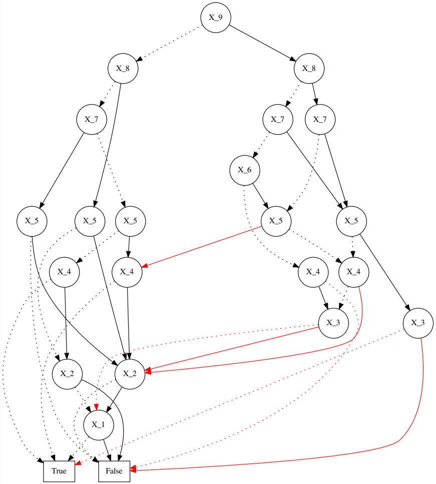
  spanning tree given by a depth-first search

  - ⇒ tree edges
  - non-tree edges

Iterative counting

Recursive versus iterative counting

In my CLA'20 talk [Latin'20]: **recursive counting**



Recursive versus iterative counting

In my CLA'20 talk [Latin'20]: **recursive counting**

The time complexity (in the number of arithmetic operations) for partitioning the Boolean functions in k variables according to their ROBDD size is

$$\Omega(M_k^{1.5 \cdot \log M_k}),$$

where M_k if the largest ROBDD on k variables.

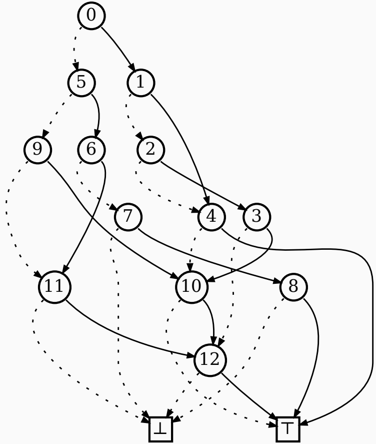
$$(M_k)_{k=1,\ldots,12} = (3, 5, 7, 11, 19, 31, 47, 79, 143, 271, 511, 767).$$

We have $\dfrac{2^k}{k} \leq M_k \leq 2 \cdot \dfrac{2^k}{k}$ as k tends to infinity.

In practice, we computed the partition for $k = 8$ in about 2 hours on a personal computer with a python implementation
and for $k = 9$, in several days using a fast computer with a C++ implementation.

Today: **iterative counting**

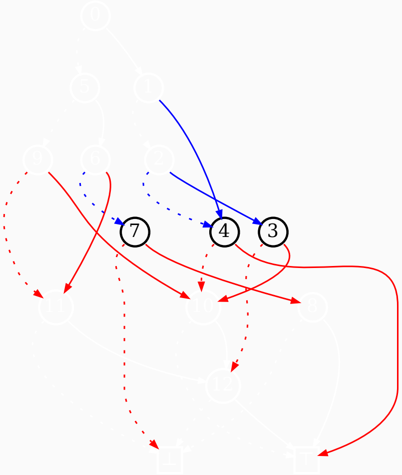
through inclusion-exclusion principle



Recursive versus iterative counting

Today: **iterative counting**

through inclusion-exclusion principle



- 7 incoming edges
- 3 nodes
- 9 outgoing edges

Recursive versus iterative counting

Today: **iterative counting**

through inclusion-exclusion principle

The time complexity (in the number of arithmetic operations) for partitioning the Boolean functions in k variables according to their ROBDD size is

$$O(M_k^4 \cdot \log M_k),$$

where M_k if the largest ROBDD on k variables.

$$(M_k)_{k=1,\dots,12} = (3, 5, 7, 11, 19, 31, 47, 79, 143, 271, 511, 767).$$

We have $M_k \approx 2^k/k$ as k tends to infinity.

In practice, we compute the partition for $k = 12$, thus for the 2^{4096} functions in about 6 hours on a computer with > 200GB of RAM.

cf. `https://github.com/agenitrini/BDDgen`

Unranking a ROBDD

The *Unranking* method for combinatorial structures

1. Defining a total order over the structures
   (same number of Boolean variables and same size).
2. Constructing the structure only by using its **rank**.

$\Rightarrow$ This allows to build structures of a given size.

$\Rightarrow$ This leads trivially to a uniform (by size) random sampler.

The *Unranking* algorithm

`Inputs:` rank `r`, size `s`, number of variables `k`

1. Select the profile;

   Done by iteratively computing the number of nodes by layer.

The *Unranking* algorithm

`Inputs:` rank `r`, size `s`, number of variables `k`

1. Select the profile;

2. Build the spine;

   Reverse postorder traversal gives information about non-tree edges.

The *Unranking* algorithm

`Inputs:` rank `r`, size `s`, number of variables `k`

1. Select the profile;

2. Build the spine;

3. Build the ROBDD;
   Inorder traversal allows to decide the destination of non-tree edges.

The *Unranking* algorithm

`Inputs:` rank `r`, size `s`, number of variables `k`

1. Select the profile;
2. Build the spine;
3. Build the ROBDD;

The unranking algorithm with inputs r, n, k satisfies:

- $O(k^2\ n^5)$ time complexity and $O(n^2)$ extra space for identifying the profile;
- $O(k^2\ n^3)$ time complexity to generate the ROBDD with a given profile.

The *Unranking* algorithm

`Inputs:` rank `r`, size `s`, number of variables `k`

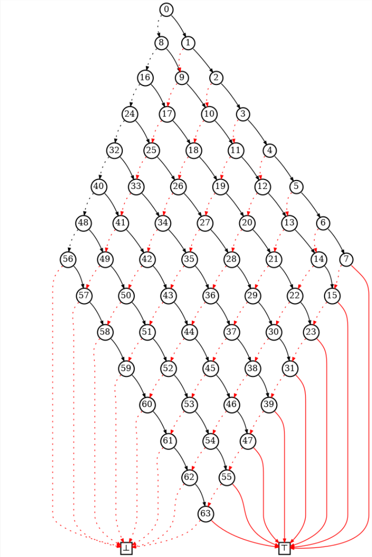
1. Select the profile;
2. Build the spine;
3. Build the ROBDD;

The unranking algorithm with inputs r, n, k satisfies:

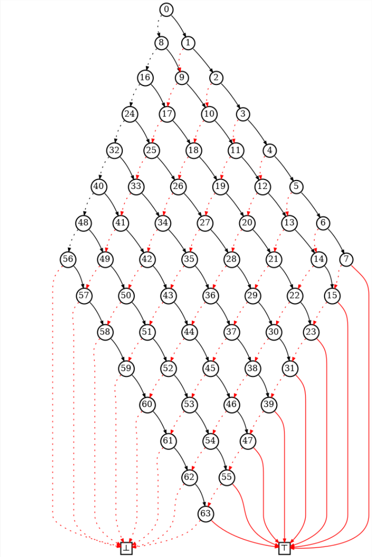
- $O(k^2\ n^5)$ time complexity and $O(n^2)$ extra space for identifying the profile;
- $O(k^2\ n^3)$ time complexity to generate the ROBDD with a given profile.

We can shortcut Step 1, or Step 1 and 2.

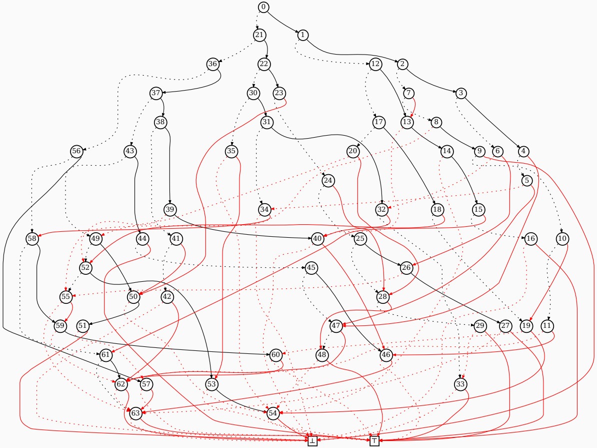
ROBDD gallery: Majority with 15 variables



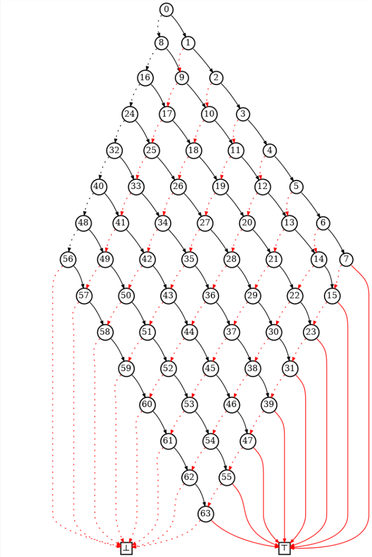
ROBDD gallery: Majority with 15 variables



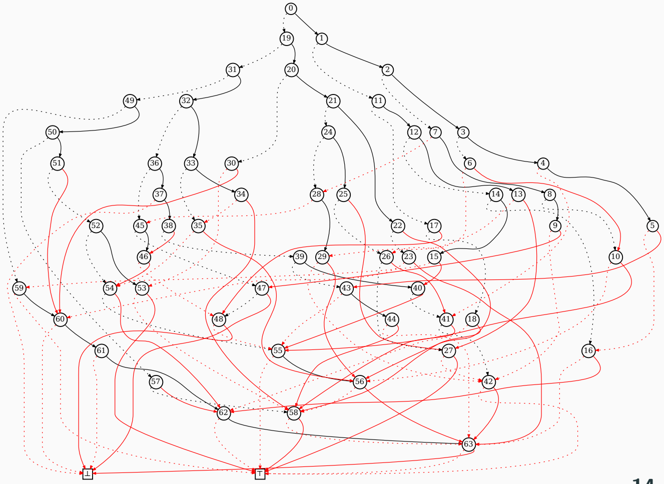
Same size as Majority with 15 variables.



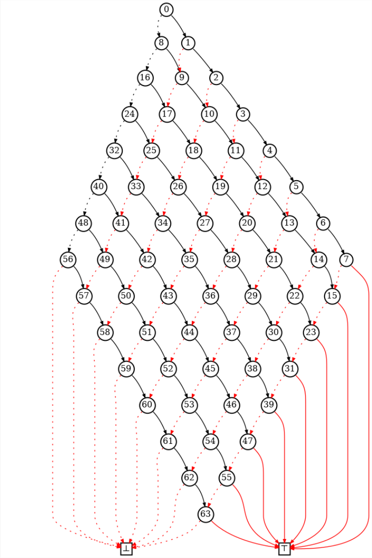
ROBDD gallery: Majority with 15 variables



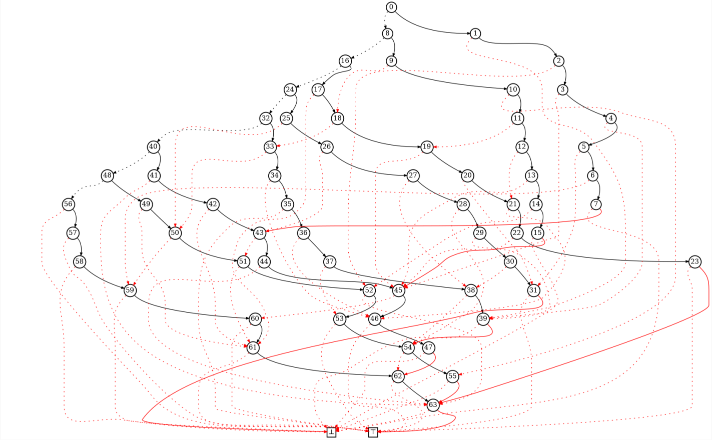
Same profile as Majority with 15 variables.



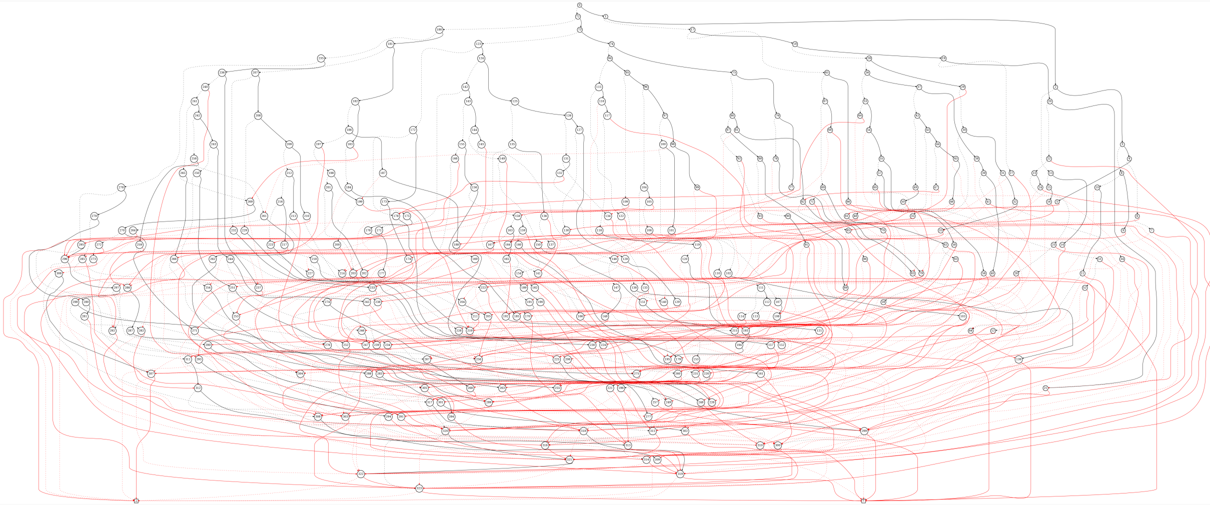
ROBDD gallery: Majority with 15 variables



Same spine as Majority with 15 variables.



ROBDD gallery: same spine as Majority with 35 variables



Conclusion and future work

– Our combinatorial approach adapts very well.

- subclasses of functions: ex. only with essential variables
- other classes of BDDs (with other constraint rules)
    - OBDDs
    - Quasi-reduced BDDs
    - Zero-supressed BDDs

– In the future:

- Combinatorial characterization of the class of spines?
- Can we improve the time complexity for the counting/unranking methods?