Are Pigeon Hole and Urquhart Problems really exponential for DP?

Laurent Simon

LRI – Université Paris-Sud

Joint work with Philippe Chatalic

SAT2000 Workshop, Renesse

Plan

- Original DP procedure
- Efficient data structure (BDDs):
  - Suited to structured problems (*Hypothesis*: They should show some regularities)
- Experimental results
  - Pigeon Hole, Urquhart, Dimacs

Exponential size formula can be handled by DP

The Davis-Putnam Procedures

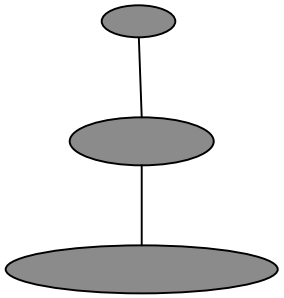
DP (1960)

```
func DP(f, V):

  If V = {} return f
  x = choose_var(f, V)
  return
       DP(f_{x<-1} ∨ f_{x<-0} ,V\{x})
```

(Distribution)

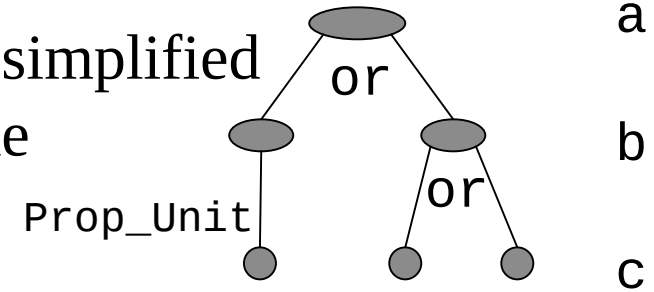
Sequence of resolvents



DLL (1962)

```
func DLL(f, V):
  Prop_Unit(f, V)
  If V = {} return f
  x = choose_var(f, V)
  return
        DLL(f_{x<-1} , V\{x}
    OR DLL(f_{x<-0} , V\{x})
```

Tree of simplified formulae



Previous Implementations of the original DP procedures

- DR (*Dechter-Rish-94*, *Rish-99*)
  - May succeed when DLL fails
  - Importance of the structure of instances
  - Induced width (w*)
- CalcRes (*Chatalic-Simon-99*)
  - Resolvent calculus
  - Extensive DP experimentation

Previous Results of *classical* DP procedures

- Bad results on random instances
- Performs better on *structured* instances

- Major problem:
  - Size of sets of clauses

How to efficiently encode large formulas?

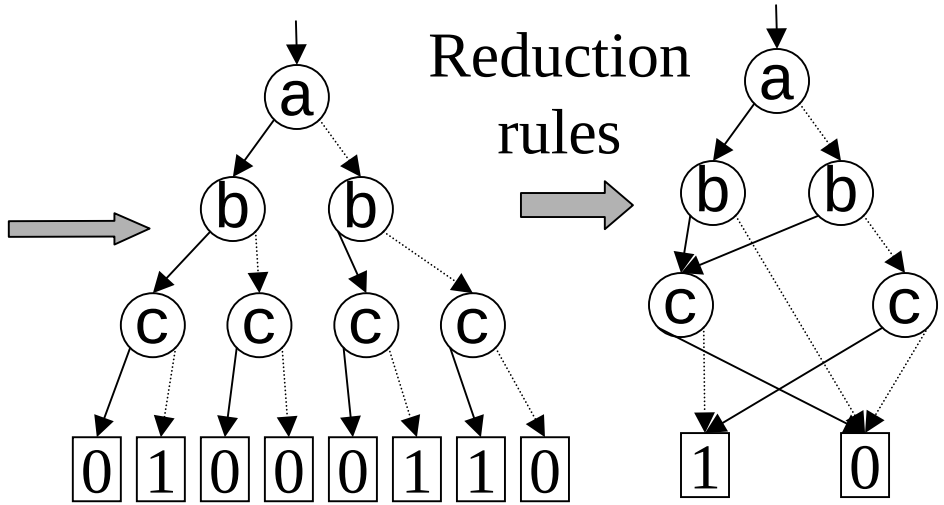
BDDs *(Bryant-86)*

- Efficient encoding of boolean functions
  - Exploit regularities to compress them
- Effective with *real-world* problems
  - Fault-Tree diagnosis
  - PI *(Madre-Coudert-90)*
  - ...

Binary Decision Diagrams

$(a \vee b \vee c) \wedge (\neg a \vee b) \wedge (\neg b \vee \neg c)$

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



Reduction rules

- Shannon Decomposition
- BDDs represents all models

  ⇒ More difficult than SAT

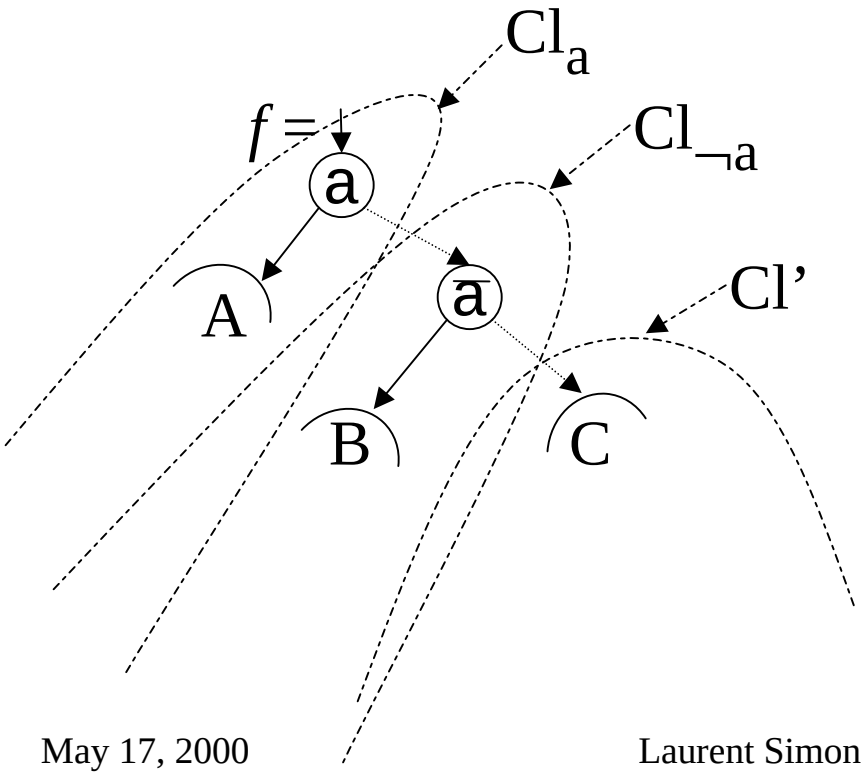
BDDs as a Data Structure

- Zero-suppressed BDDs (*Minato-96*)
  - Change one reduction rule of BDDs
- Avoid computing Shannon Normal Form
- Encodes set of clauses
- Operations only depend on ZBDDs Size
  - Not on the size of the encoded set

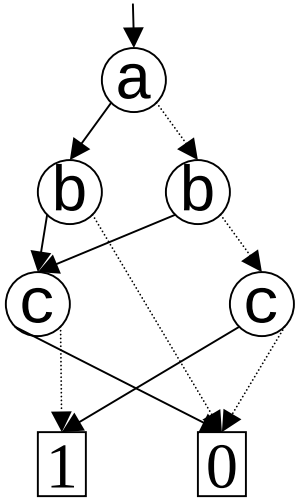
Example of encoded formula

$Cl = \mathrm{Cl}_a \cup \mathrm{Cl}_{\neg a} \cup \mathrm{Cl}'$

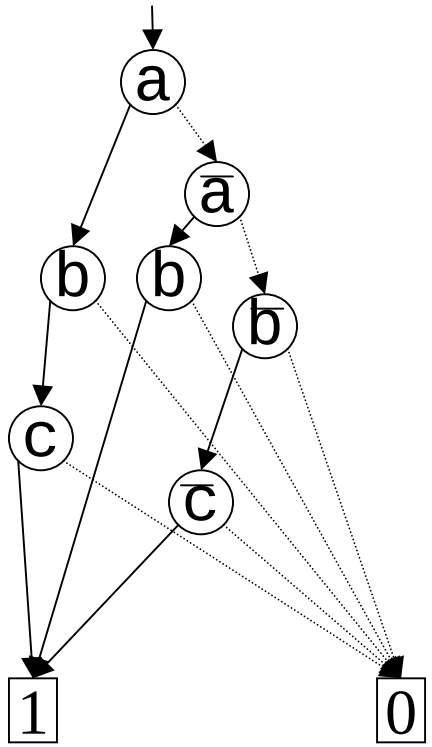
$f = (a \vee A) \wedge (\neg a \vee B) \wedge C$



$(a \vee b \vee c) \wedge (\neg a \vee b) \wedge (\neg b \vee \neg c)$



SNF



Clauses

ZRes

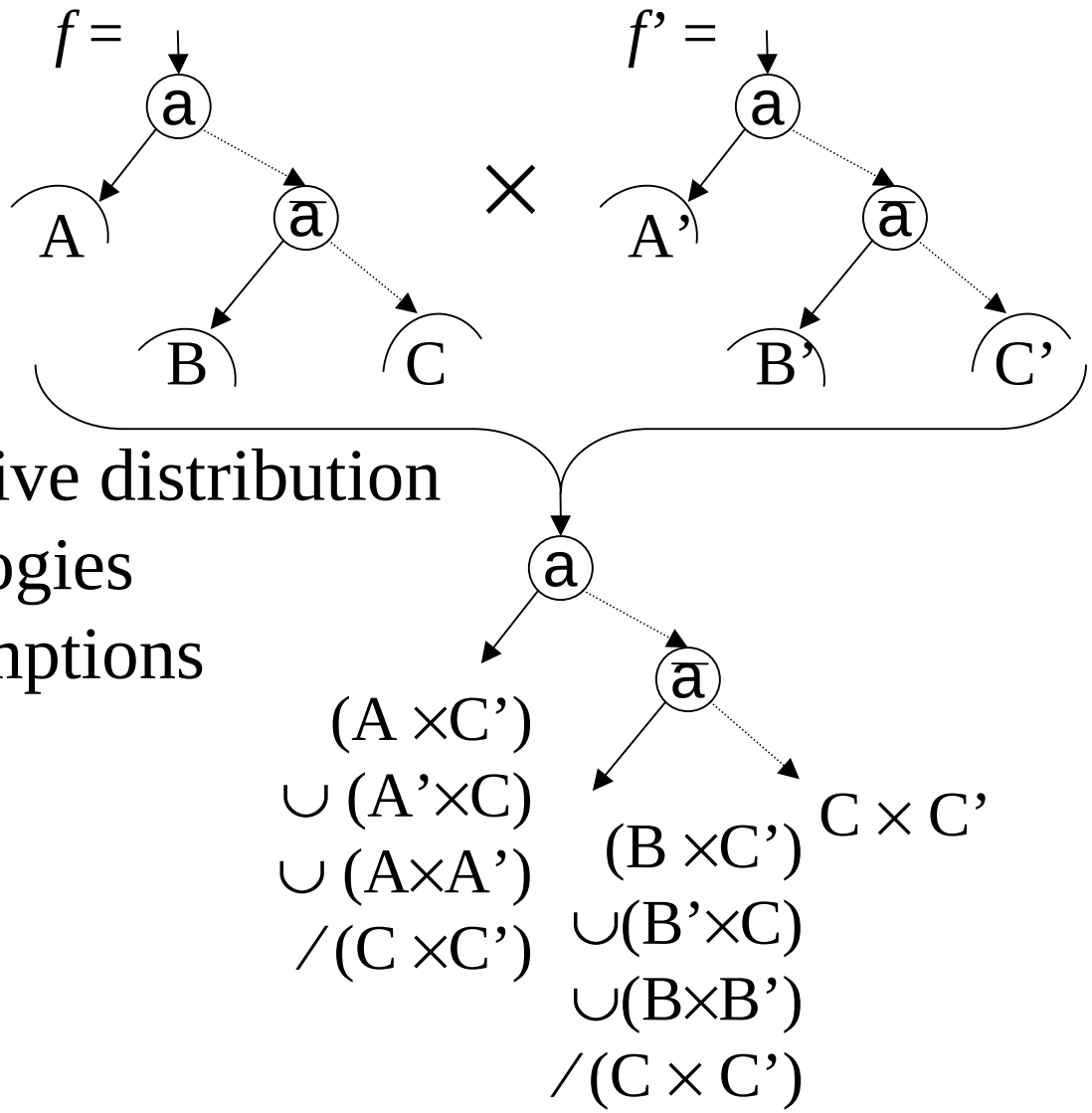
- Handle set of clauses by ZBDDs encoding
- Efficient implementation of DP
- Distribution implemented at the set level
  - Independent from the number of clauses encoded in the set
  - “On the fly”
    - Subsumed clauses deletions
    - Tautologies deletions

ZRes (2)

- Implemented in C, using the CUDD package (*Somenzi*)
- Dynamic heuristics choices
- Dynamic ZBDDs variables ordering

- 2 versions:
  - Standard: classical distribution operator
  - Clause-distribution: specialized operator

The Distribution Operator ($\times$)



- Recursive distribution
- Tautologies
- Subsumptions

Experiments

- Pigeon Hole and Urquhart problem
- Why these benchmarks?
  - Importance in complexity
  - Should generate large resolvents
- Comparison with:
  - Asat (*Dubois*)
  - Sato (*Zhang*)

The Pigeon Hole Principle

"n+1 pigeons can't sit in n holes"

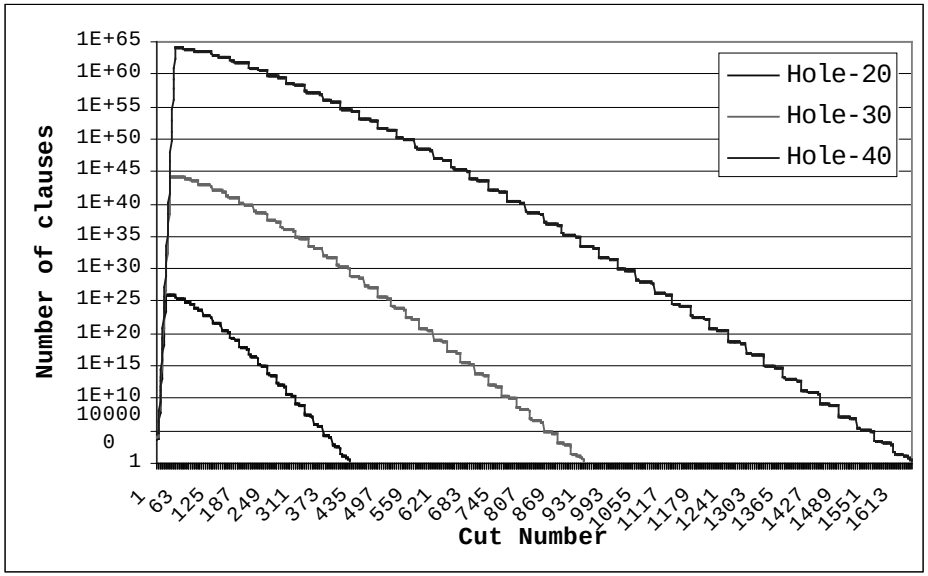
- First problem proved hard for resolution *(Haken-85)*

- DLL Comparison *(time in seconds on a PII-400)*

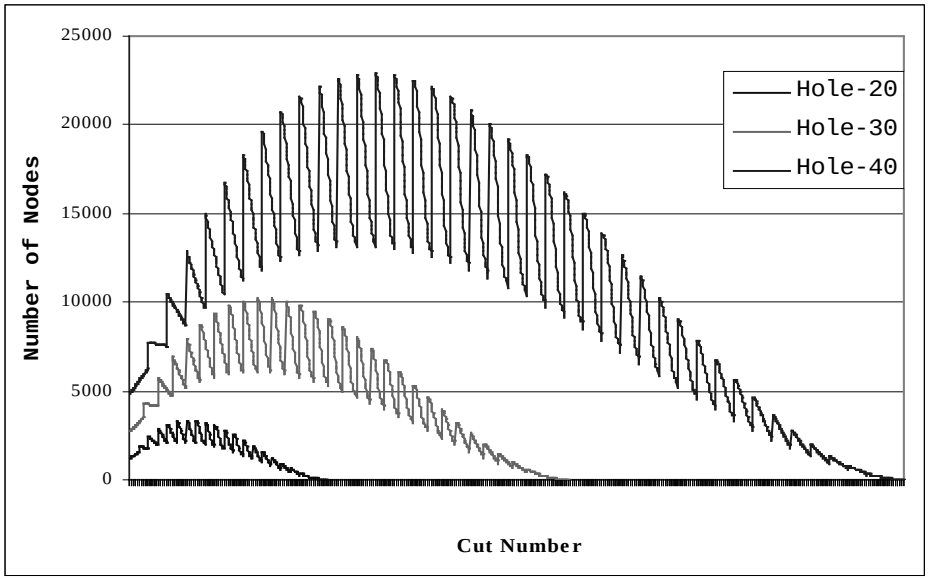
Instances	Asat	Sato	ZRes(std)	ZRes (cd)
Hole-09	11.94	8.90	1.87	1.01
Hole-10	141.96	80.94	3.26	1.61
Hole-11	1960.77	7373.65	5.76	2.65
Hole-12	10000	10000	10.18	4.06
Hole-20	--	--	654.8	69
Hole-30	--	--	10000	1102
Hole-40	--	--	--	9421

Pigeon Hole

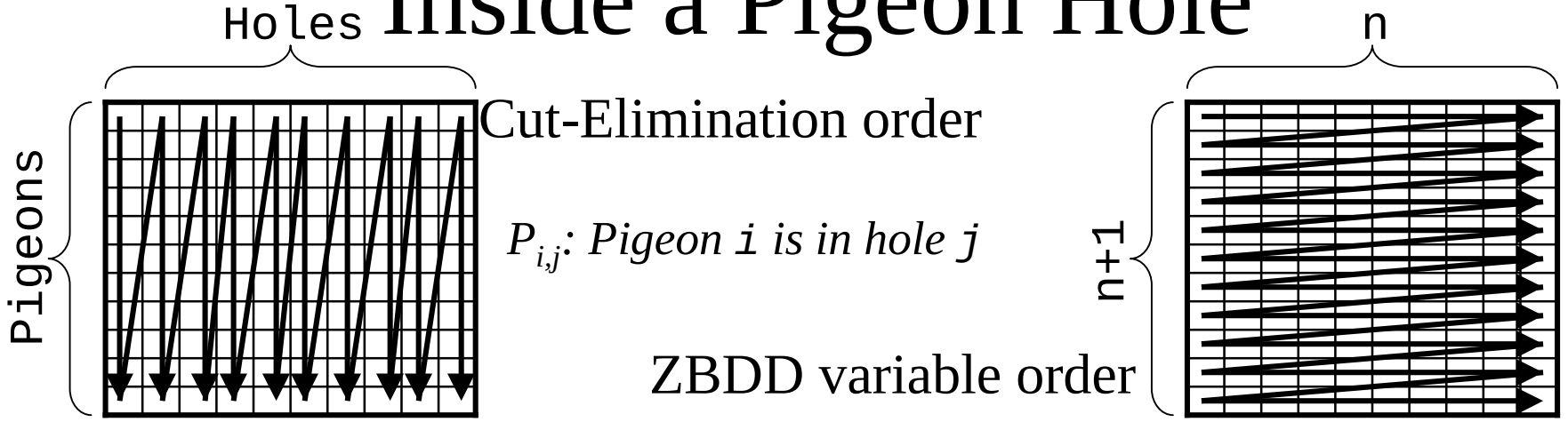
- Clauses:
  1. Exponential growth
  2. Regular decrease



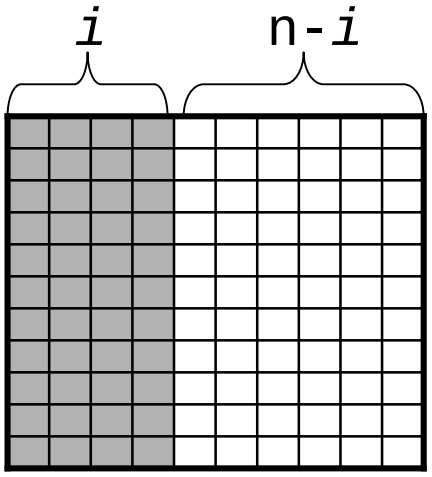
- Nodes:
  - Periodic phenomenon
  - Polynomially bounded



Inside a Pigeon Hole



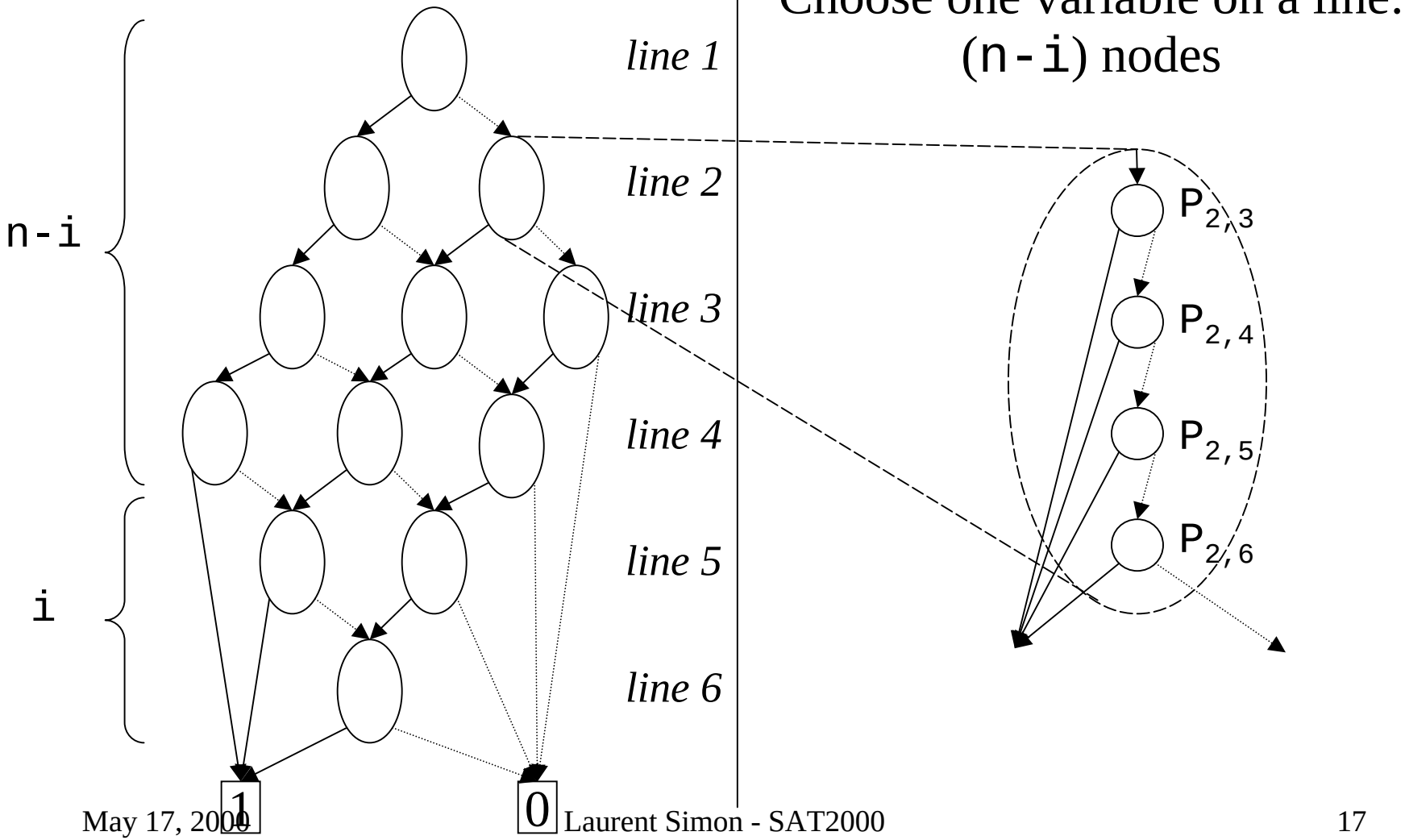
Every `(n+1)` cut, the formula contains all possible clauses of length `(n-i)` that contains one positive literal of each lines



Number of clauses: $\binom{n+1}{n-i}(n-i)^{(n-i)}$

Pigeon Hole: Number of nodes

Choose (n-i) lines (n=6, i=2)



Urquhart Problems

- Second problem used to prove weakness of resolution
- Encodes a property of expander graphs
  - Tseitin-68, Galil-77, Urquhart-87
- Contains equivalency or exclusives clauses
  - $a \leftrightarrow b \leftrightarrow c \leftrightarrow d \leftrightarrow e$
  - $a \oplus b \oplus c \oplus d \oplus e$

Results on Urquhart's problems

- Comparison with DLL provers *(Urq-3, 100 exp.)*:

System	Total cpu time	# Solved	Mean cpu time (solved)
Asat	404287	69	1366
Sato	776364	26	1398
ZRes (cd)	69.2	100	0.69

- ZRes on much bigger instances *(1000 exp.)*:

(Mean values)

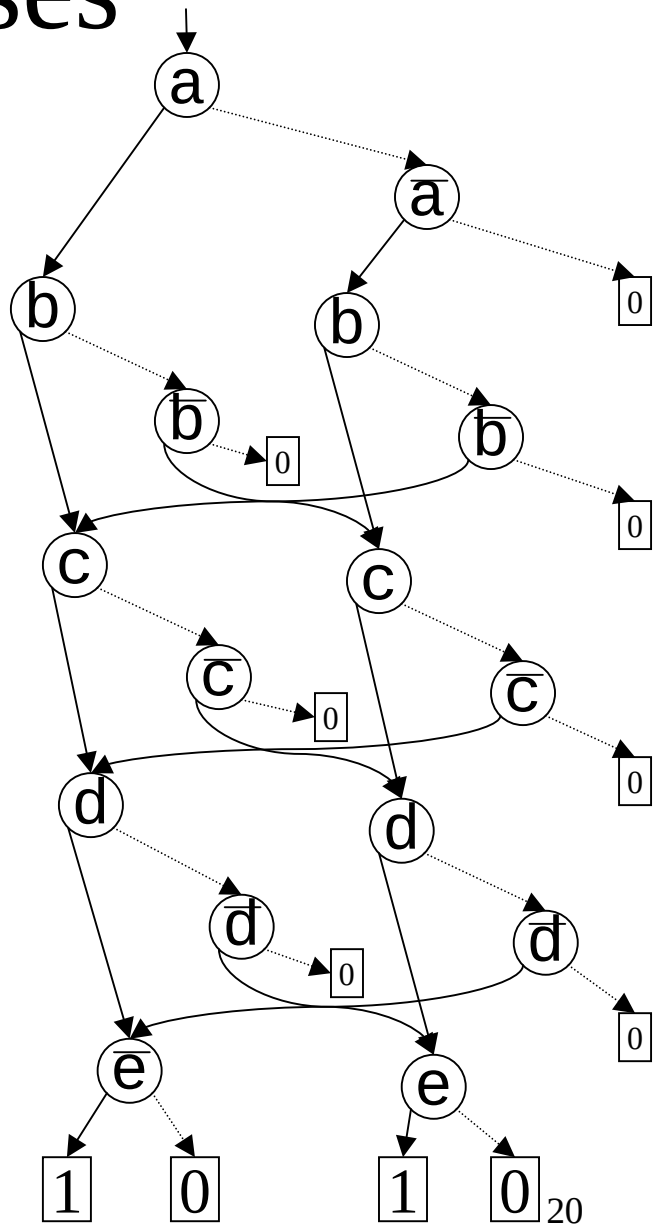
Urq	Initial Nb. Var.	Initial Nb. Cl.	Max Nb. Cl.	Max. Nb. Nodes	ZRes Time
4	78	714	10^10	440	1.72
6	178	1683	10^24	1011	8.88
8	318	3024	10^43	1808	29.6
10	498	4754	10^67	2833	72.0

Equivalency clauses

$a \leftrightarrow b \leftrightarrow c \leftrightarrow d \leftrightarrow e$

Clauses containing an odd/even number of negated literals

$abcd\bar{e}$, $abc\bar{d}e$, $ab\bar{c}de$, $a\bar{b}cde$, $\bar{a}bcde$,
$\bar{a}bcd\bar{e}$, $\bar{a}bc\bar{d}e$, $\bar{a}b\bar{c}de$, $\bar{a}\bar{b}cde$, $\bar{a}\bar{b}c\bar{d}e$,
$\bar{a}\bar{b}\bar{c}de$, $a\bar{b}c\bar{d}\bar{e}$, $a\bar{b}\bar{c}d\bar{e}$, $a\bar{b}\bar{c}\bar{d}e$, $ab\bar{c}\bar{d}\bar{e}$,
$\bar{a}\bar{b}\bar{c}d\bar{e}$

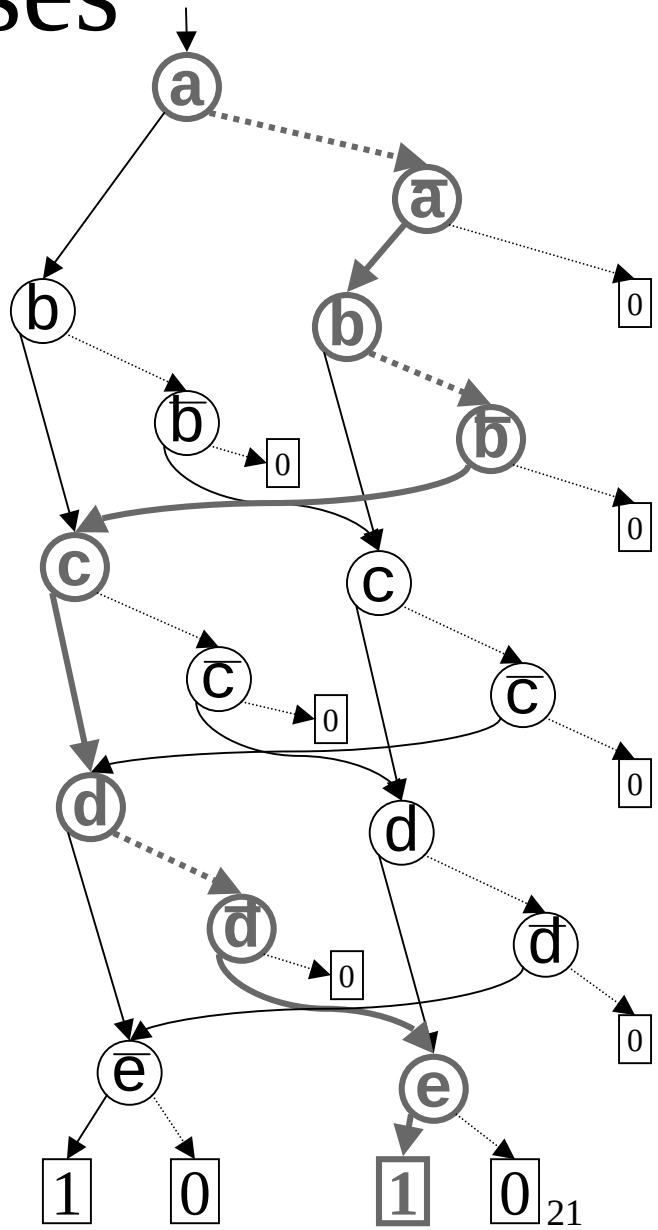


Equivalency clauses

$a \leftrightarrow b \leftrightarrow c \leftrightarrow d \leftrightarrow e$

Clauses containing an odd/even number of negated literals

$abcd\bar{e}$, $abc\bar{d}e$, $ab\bar{c}de$, $a\bar{b}cde$, $\bar{a}bcde$,
$\bar{a}bc\bar{d}\bar{e}$, $\bar{a}b\bar{c}\bar{d}e$, $\bar{a}b\bar{c}d\bar{e}$, $\bar{a}\bar{b}cd\bar{e}$, $\bar{a}\bar{b}\bar{c}de$,
$\bar{a}\bar{b}\bar{c}\bar{d}\bar{e}$, $\bar{a}\bar{b}cd\bar{e}$, $\bar{a}b\bar{c}d\bar{e}$, $\bar{a}b\bar{c}\bar{d}e$, $a\bar{b}\bar{c}\bar{d}e$, $ab\bar{c}\bar{d}\bar{e}$,
$\bar{a}\bar{b}\bar{c}\bar{d}\bar{e}$



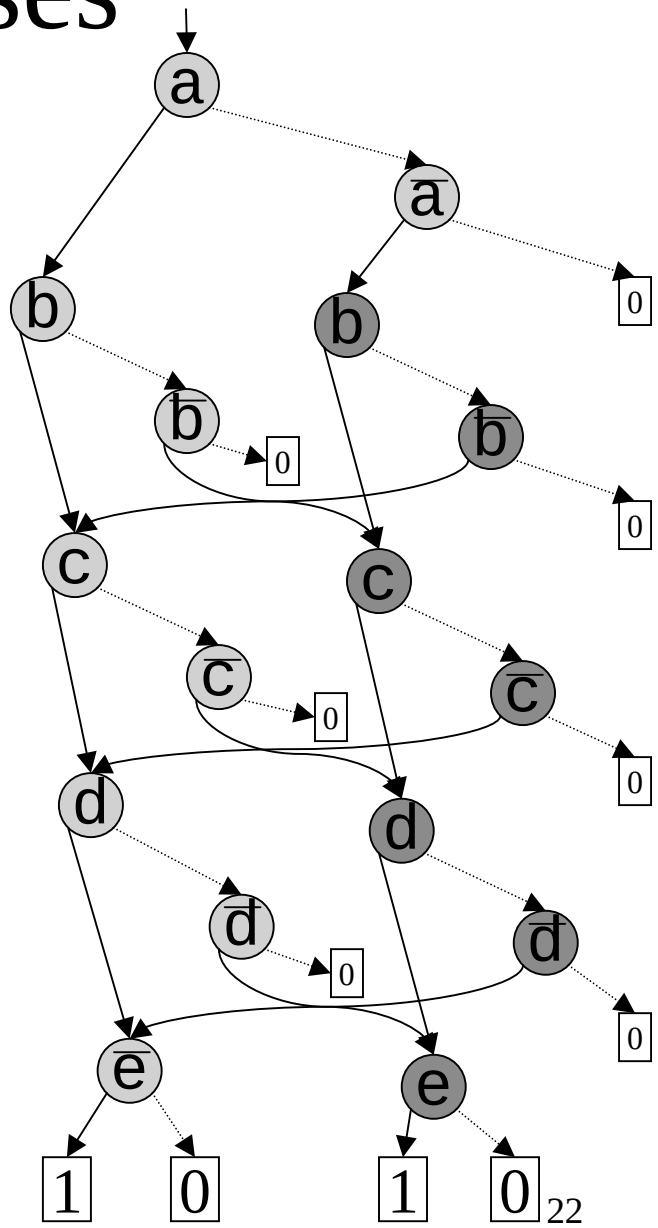
Equivalency clauses

$a \leftrightarrow b \leftrightarrow c \leftrightarrow d \leftrightarrow e$

Clauses containing an odd/even number of negated literals

$abcd\bar{e}$, $abc\bar{d}e$, $ab\bar{c}de$, $a\bar{b}cde$, $\bar{a}bcde$, $\bar{a}bcd\bar{e}$, $\bar{a}bc\bar{d}e$, $\bar{a}b\bar{c}de$, $\bar{a}\bar{b}cde$, $\bar{a}\bar{b}\bar{c}de$, $\bar{a}\bar{b}c\bar{d}e$, $\bar{a}b\bar{c}\bar{d}e$, $a\bar{b}\bar{c}\bar{d}e$, $\bar{a}\bar{b}\bar{c}\bar{d}\bar{e}$

- n variables
- 4n-2 Nodes
- $2^{(n-1)}$ Clauses



Conclusion

- Well suited to encode highly structured instances
- Outperforms DLL on this instances
- And on other examples?
  - Dimacs: Sato is still better
  - Are Pigeon Hole and Urquhart too regular?
  - *Real-world* instances generally have particular structure

Measuring compression level

- Size of Formula $(\sum_{i=1}^{nbc} length(C_i))$/ Number of nodes
- Observation of compression variation

Problem	Initial	Maximal
Alea-130c-30v	2.06 *(0.02)*	4.58 *(0.34)*
Alea-180c-42v	2.05 *(0.02)*	6.15 *(0.44)*
Pret150	1.88 *(0)*	1157*(0)*
Par32-c	2.14*(0.02)*	10^7 *(10^7)*
Flat-50	1.76*(0.01)*	9.61 *(2.14)*
Ssa	1.72 *(0.01)*	5.72 *(0.28)*
Massacci-R4	7.16 *(0.25)*	72.9 *(8.95)*
Aralia	1.79 *(0.28)*	10^6 *(10^7)*

Measuring compression level

- Size of Formula $(\sum_{i=1}^{nbc} length(C_i))$/ Number of nodes
- Observation of compression variation

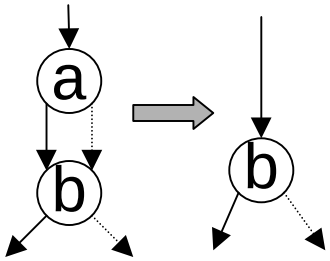
Problem	Entry	Best
Alea-130c-30v	**2.06 *(0.02)***	**4.58 *(0.34)***
Alea-180c-42v	**2.05 *(0.02)***	**6.15 *(0.44)***
Pret150	1.88 *(0)*	1157*(0)*
Par32-c	2.14*(0.02)*	10^7 *(10^7)*
Flat-50	**1.76*(0.01)***	**9.61 *(2.14)***
Ssa	**1.72 *(0.01)***	**5.72 *(0.28)***
Massacci-R4	7.16 *(0.25)*	72.9 *(8.95)*
Aralia	1.79 *(0.28)*	10^6 *(10^7)*

Random instances

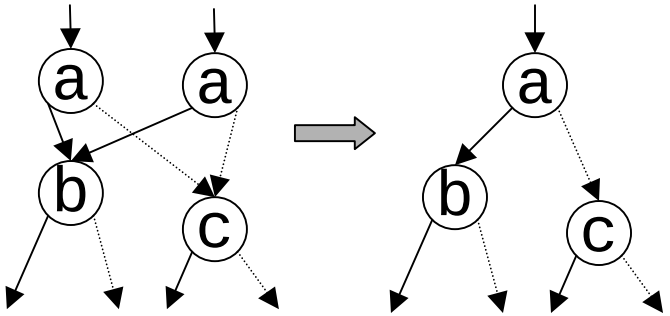
Conclusion – (2)

What is this proof system?

BDDs reduction rules

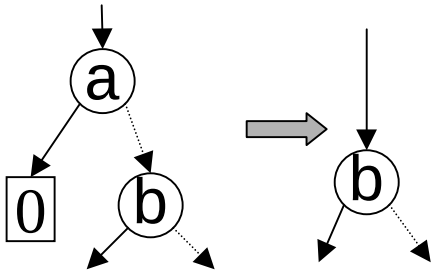


Node-Elimination



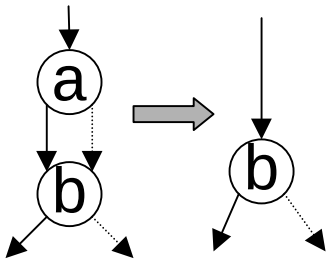
Node-Sharing

ZBDDs reduction rules



Z-Elimination

Replaces



Node-Elimination

But: In a clause-set context, the Node-Elimination rule is preserved as the subsumption rule… Good point for ZBDD...

Some details on Pigeon Hole if i columns are deleted:

- Number of clauses *(approx.)*:

$$\binom{n-i+1}{n-1}.(n-i+1)^{(n-i+1)}$$

- Number of Nodes:

$$2i^3 - i^2.(4n+2) + i(2n^2 - n + 3) + 3n^2 - n - 3$$