SAT encodings: using the right tool for the right job

Ruben Martins

University of Oxford

April 2, 2015

How to encode a problem into SAT?

```
c famous problem (in CNF)
p cnf 6 9
1 4 0
2 5 0
3 6 0
-1 -2 0
-1 -3 0
-2 -3 0
-4 -5 0
-4 -6 0
-5 -6 0
```

How to encode a problem into SAT?

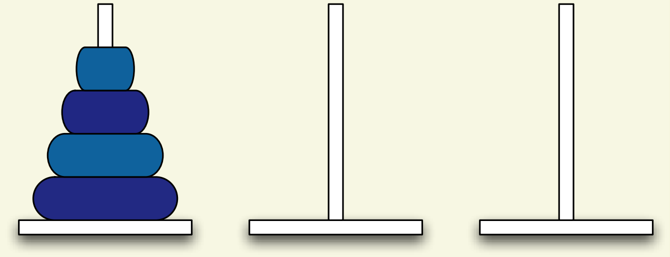
```
c pigeon hole problem
p cnf 6 9
1 4 0          # pigeon[1]@hole[1] ∨ pigeon[1]@hole[2]
2 5 0          # pigeon[2]@hole[1] ∨ pigeon[2]@hole[2]
3 6 0          # pigeon[3]@hole[1] ∨ pigeon[3]@hole[2]
-1 -2 0        # ¬pigeon[1]@hole[1] ∨ ¬pigeon[2]@hole[1]
-1 -3 0        # ¬pigeon[1]@hole[1] ∨ ¬pigeon[3]@hole[1]
-2 -3 0        # ¬pigeon[2]@hole[1] ∨ ¬pigeon[3]@hole[1]
-4 -5 0        # ¬pigeon[1]@hole[2] ∨ ¬pigeon[2]@hole[2]
-4 -6 0        # ¬pigeon[1]@hole[2] ∨ ¬pigeon[3]@hole[2]
-5 -6 0        # ¬pigeon[2]@hole[2] ∨ ¬pigeon[3]@hole[2]
```

Encoding to CNF

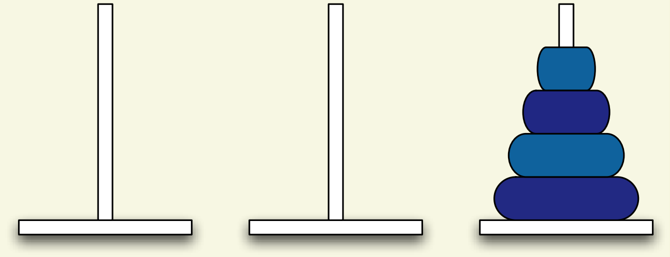
- What to encode?
  - Boolean formulas
  - Cardinality constraints
    - $x_1 + \ldots + x_n \leq k$
  - Arithmetic
    - Addition, Comparison, Multiplication...
  - ...

- Which encoding to use?
  - Different encodings have a major impact on performance !

Encoding a problem into SAT – Towers of Hanoi



Encoding a problem into SAT – Towers of Hanoi



- Only one disk may be moved at a time;
- No disk may be placed on the top of a smaller disk;
- Each move consists in taking the upper disk from one of the towers and sliding it onto the top of another tower.

How to encode ToH?

STRIPS planning mode:

- Variables
- Actions: preconditions $\rightarrow$ postconditions
- Initial state
- Goal state

How to encode ToH?

[Selman & Kautz ECAI'92]

- Variables: $on(d, dt, i)$; $clear(dt, i)$
- Actions: $move(d, dt, dt, i) = obj(d, i) \wedge from(dt, i) \wedge to(dt, i)$
  - preconditions:
    $clear(d, i), clear(dt', i), on(d, dt, i)$
  - postconditions:
    $on(d, dt', i+1), clear(dt, i+1), \neg on(d, dt, i), \neg clear(dt', i+1)$
- Initial state:
  - $on(d_1, d_2, 1), \ldots, on(d_{n-1}, d_n, 1), on(d_n, t_1, 1)$
    $clear(d_1, 1), clear(t_1, 1), clear(t_2, 1), clear(t_3, 1)$
  - All other variables initialized to false
- Goal state:
  - $on(d_1, d_2, 2^n - 1), \ldots, on(d_{n-1}, d_n, 2^n - 1), on(d_n, t_1, 2^n - 1)$

How to encode ToH?

[Selman & Kautz ECAI'92]

Constraints:

- Exactly one disk is moved at each time step
- There is exactly one movement at each time step
- There are no movements to exactly the same position
- For a movement to be done the preconditions must be satisfied
- After performing a movement the postconditions are implied
- No disks can be moved to the top of smaller disks
- Initial state holds at time step 0
- Goal state holds at time step $2^n - 1$
- Preserve the value of variables that were unaffected by movements

How good is this encoding?

Time limit of 10,000 seconds using **picosat**

n	Selman
4	0.16
5	8.31
6	54.70
7	5252.27
8	-
9	-
10	-
11	-
12	-

A more compact encoding

[Prestwich SAT'07]

- Actions: $move(d, dt, dt, i) = obj(d, i) \wedge from(dt, i) \wedge to(dt, i)$
  - Before:
    - Movements from disks/towers to disks/towers
  - Now:
    - Movements from towers to towers
    - Clear variable can be removed

- More compact encoding:
  - Before: 5 towers requires 1,931 variables and 14,468 clauses
  - Now: 5 towers only requires 821 variables and 6,457 clauses

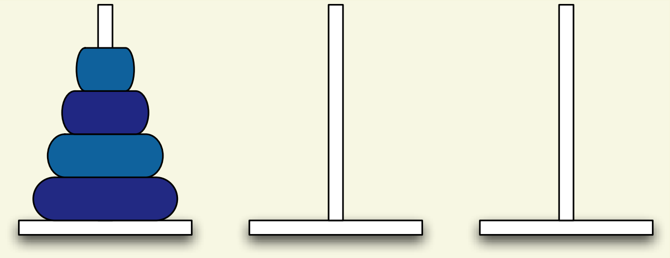
How good is this encoding?

n	Selman	Prestwich
4	0.16	0.01
5	8.31	0.08
6	54.70	0.47
7	5252.27	3.65
8	-	109.7
9	-	7126.57
10	-	-
11	-	-
12	-	-

- Can we do better?
  - Look at the properties of the problem **!**

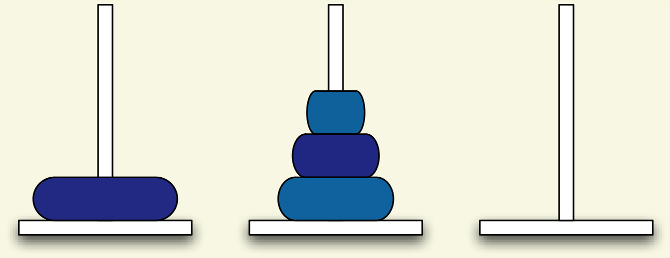
[Martins & Lynce LPAR'08]

ToH Properties (Recursion)



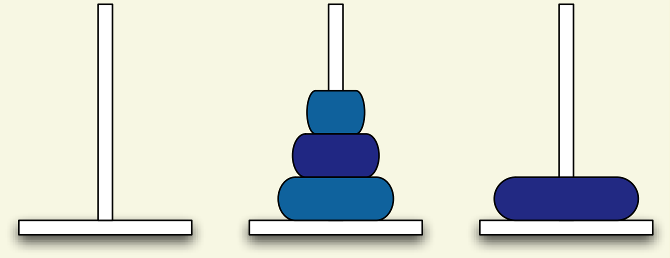
- Given a ToH of size n, one may easily find a solution taking into account the solution for a ToH of size $n - 1$

ToH Properties (Recursion)



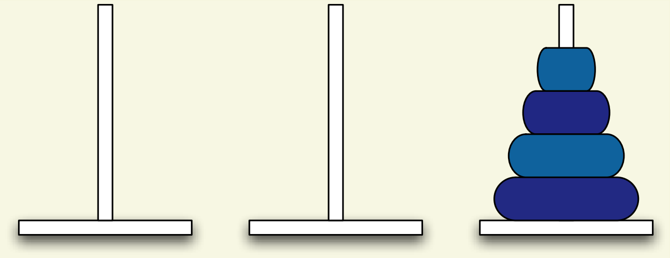
- Given a ToH of size n, one may easily find a solution taking into account the solution for a ToH of size $n-1$

ToH Properties (Recursion)



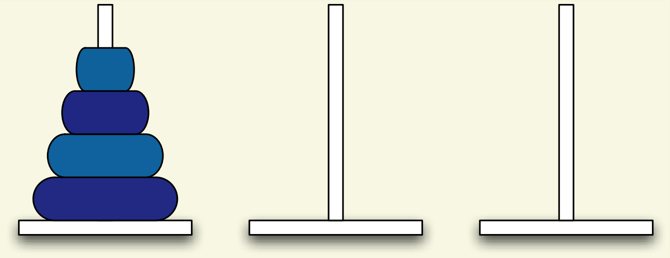
- Given a ToH of size n, one may easily find a solution taking into account the solution for a ToH of size $n - 1$
- The order of the disks to be moved after moving the largest disk is exactly the same as before

ToH Properties (Recursion)



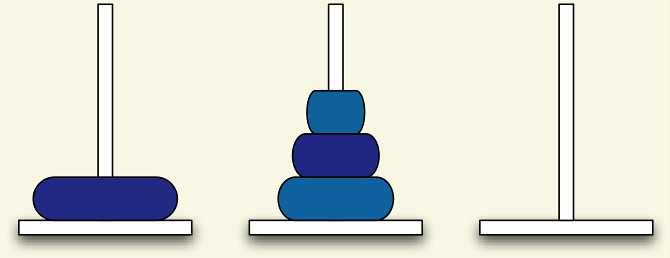
- Given a ToH of size n, one may easily find a solution taking into account the solution for a ToH of size $n - 1$
- The order of the disks to be moved after moving the largest disk is exactly the same as before

ToH Properties (Symmetry)



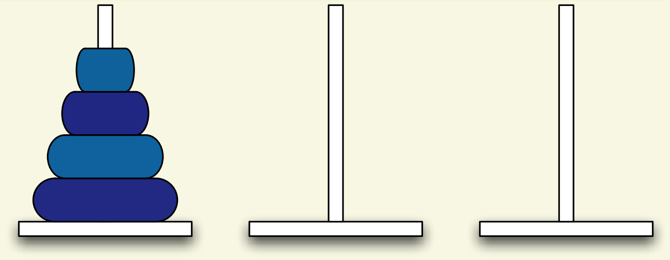
- ToH can be solved in $2^n - 1$ steps
- Considering the relationship between the movement of the disks after/before moving the largest disk we only need to determine the first $2^{n-1} - 1$ steps

ToH Properties (Symmetry)



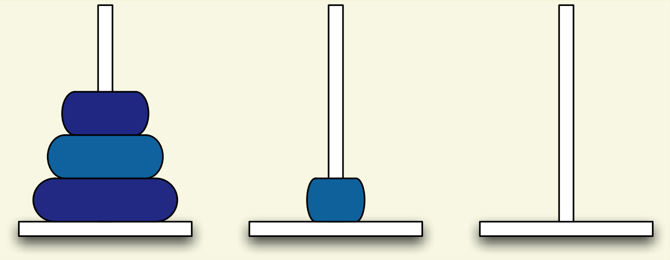
- ToH can be solved in $2^n - 1$ steps
- Considering the relationship between the movement of the disks after/before moving the largest disk we only need to determine the first $2^{n-1} - 1$ steps

ToH Properties (Parity)



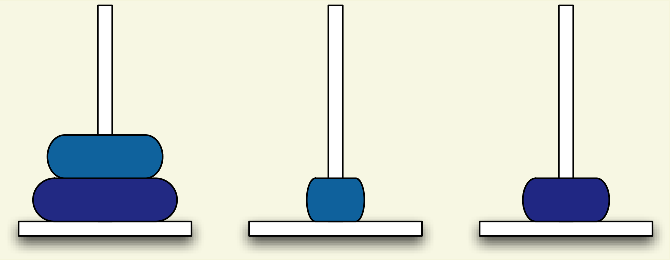
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



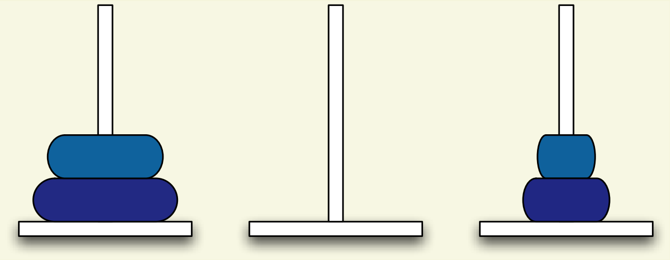
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



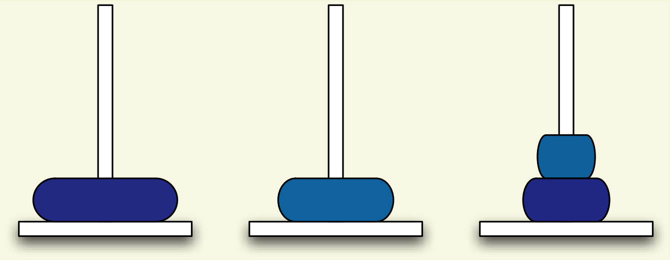
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



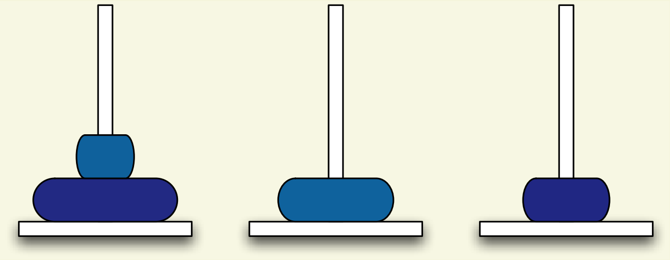
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



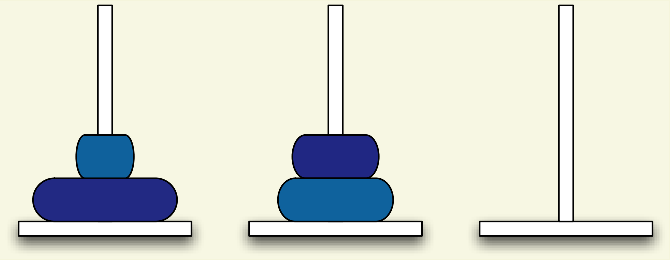
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



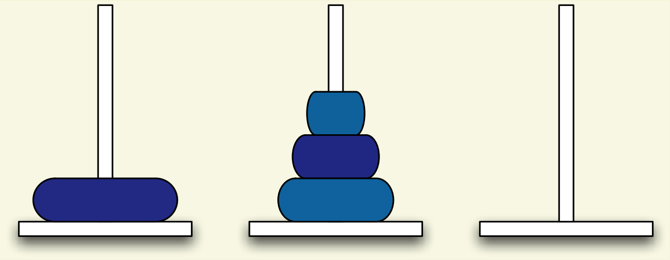
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



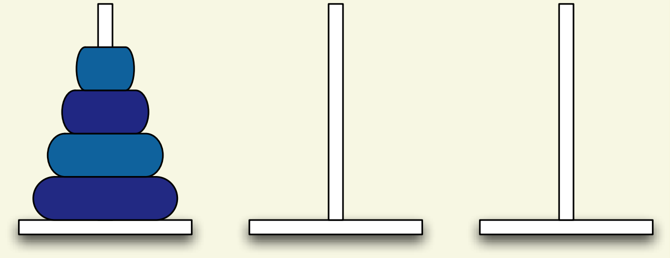
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Parity)



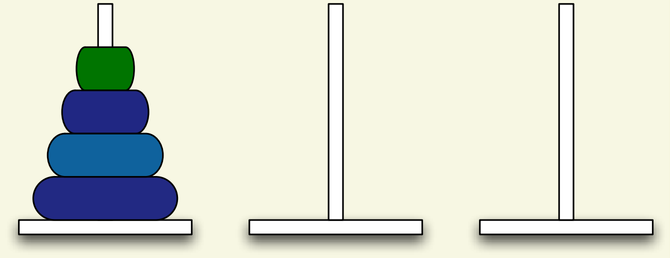
- When moving disks, no two odd/even disks can be moved next to each other

ToH Properties (Cycle)



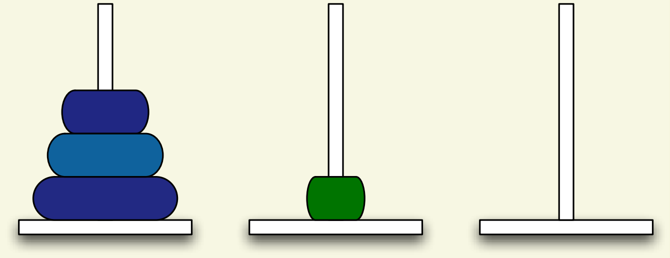
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$) while the even disks will cycle counterclockwise ($T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



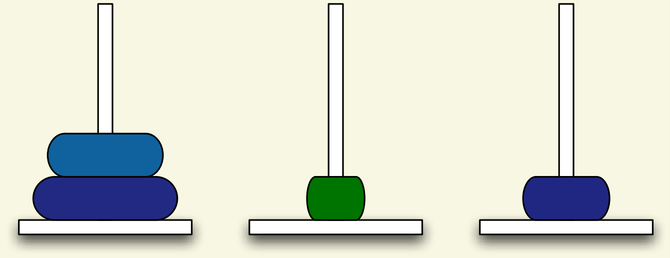
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \to T_2 \to T_3 \to T_1$) while the even disks will cycle counterclockwise ($T_1 \to T_3 \to T_2 \to T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



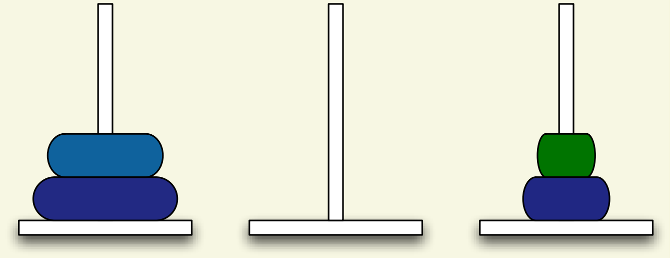
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$) while the even disks will cycle counterclockwise ($T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



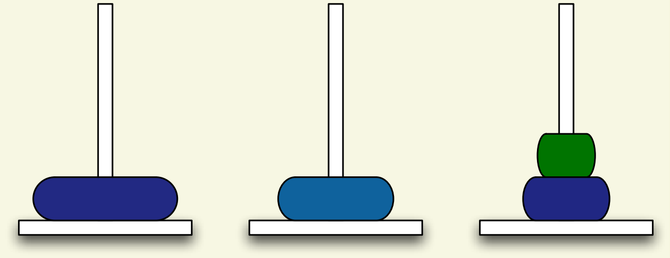
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \to T_2 \to T_3 \to T_1$) while the even disks will cycle counterclockwise ($T_1 \to T_3 \to T_2 \to T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



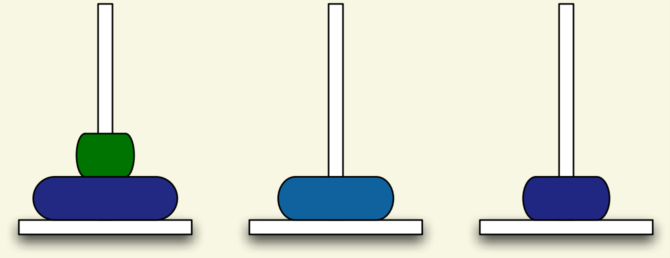
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \to T_2 \to T_3 \to T_1$) while the even disks will cycle counterclockwise ($T_1 \to T_3 \to T_2 \to T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



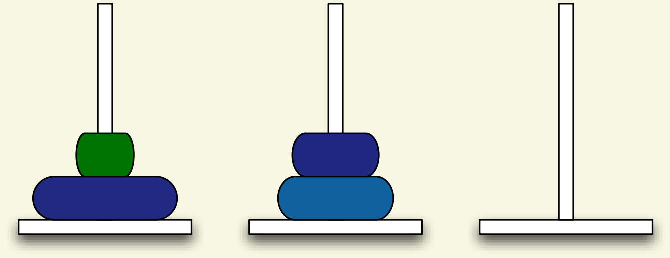
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$) while the even disks will cycle counterclockwise ($T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



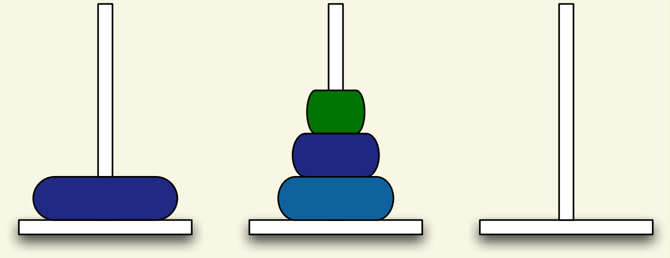
- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$) while the even disks will cycle counterclockwise ($T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \to T_2 \to T_3 \to T_1$) while the even disks will cycle counterclockwise ($T_1 \to T_3 \to T_2 \to T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

ToH Properties (Cycle)



- All disks cycle in a given order between the towers:
  - If n is even the odd disks will cycle clockwise ($T_1 \to T_2 \to T_3 \to T_1$) while the even disks will cycle counterclockwise ($T_1 \to T_3 \to T_2 \to T_1$)
  - If n is odd the odd disks will cycle counterclockwise while the even disks will cycle clockwise

Experimental Results

Size	Selman	Prestwich	Disk Parity	Disk Cycle
4	0,16	0.01	0	0
5	8.31	0.08	0.01	0.02
6	54.70	0.47	0.03	0.05
7	5252.27	3.65	0.70	0.20
8	-	109.7	5.19	5.18
9	-	7126.57	79.11	7.65
10	-	-	1997.19	973.95
11	-	-	-	1206.37
12	-	-	-	-

- Disk Parity and Disk Cycle encodings use the symmetry property

Experimental Results

Size	Selman	Prestwich	Disk Parity	Disk Cycle
4	0,16	0.01	0	0
5	8.31	0.08	0.01	0.02
6	54.70	0.47	0.03	0.05
7	5252.27	3.65	0.70	0.20
8	-	109.7	5.19	5.18
9	-	7126.57	79.11	7.65
10	-	-	1997.19	973.95
11	-	-	-	1206.37
12	-	-	-	-

- Disk Parity and Disk Cycle encodings use the symmetry property
- Can we still do better?

A new encoding for ToH

- The Disk Sequence encoding:
  - The recursive property determines the disks to be moved at each step
  - Taking into consideration this we can keep only the variables *on* and drop all the others
  - **Recursion+Symmetry+Parity**:
    - Problem can be solved with just unit propagation **!**

Unit Propagation

- Unit clause rule:
  - Given a unit clause, its only unassigned literal must be assigned value 1 for the clause to be satisfied
    - Example: for unit clause $(x_1 \vee \neg x_2 \vee \neg x_3)$, x_3 must be assigned value 0
- Unit propagation:
  - Iterated application of the unit clause rule
- Unit propagation can satisfy clauses but can also unsatisfy clauses

Experimental Results

Size	Selman	Prestwich	Disk Parity	Disk Cycle	Disk Sequence
4	0.16	0.01	0	0	0
5	8.31	0.08	0.01	0.02	0
6	54.70	0.47	0.03	0.05	0
7	5252.27	3.65	0.70	0.20	0.01
8	-	109.7	5.19	5.18	0.03
9	-	7126.57	79.11	7.65	0.09
10	-	-	1997.19	973.95	0.23
11	-	-	-	1206.37	0.56
12	-	-	-	-	1.32

Unit Propagation & Encodings

- The effect of unit propagation on encodings plays a key role on performance !

- If a fact can be derived by using only unit propagation then no search is needed !

- Which other encodings can be improved with unit propagation?
  - Cardinality constraints
  - Arithmetic operations
  - ...
  - Any encoding !

How to encode cardinality constraints?

At-most-one constraints:

- Naive (pairwise) encoding for at-most-one constraints:
  - Cardinality constraint: $x_1 + x_2 + x_3 + x_4 \leq 1$
  - Clauses:

$$\left.\begin{array}{c} (x_1 \Rightarrow \neg x_2) \\ (x_1 \Rightarrow \neg x_3) \\ (x_1 \Rightarrow \neg x_4) \\ \dots \end{array}\right\} \begin{array}{c} \neg x_1 \vee \neg x_2 \\ \neg x_1 \vee \neg x_3 \\ \neg x_1 \vee \neg x_4 \\ \dots \end{array}$$

  - Complexity: $\mathcal{O}(n^2)$ clauses

How to encode cardinality constraints?

At-most-k constraints:

- Naive encoding for at-most-k constraints:
  - Cardinality constraint: $x_1 + x_2 + x_3 + x_4 \leq 2$
  - Clauses:

$$\left.\begin{array}{c} (x_1 \wedge x_2 \Rightarrow \neg x_3) \\ (x_1 \wedge x_2 \Rightarrow \neg x_4) \\ (x_2 \wedge x_3 \Rightarrow \neg x_4) \\ \dots \end{array}\right\} \begin{array}{c} (\neg x_1 \vee \neg x_2 \vee \neg x_3) \\ (\neg x_1 \vee \neg x_2 \vee \neg x_4) \\ (\neg x_2 \vee \neg x_3 \vee \neg x_4) \\ \dots \end{array}$$

  - Complexity: $\mathcal{O}(n^k)$ clauses

Encodings for cardinality constraints

Encoding	Clauses	Variables	Type
Pairwise	$\mathcal{O}(n^2)$	0	at-most-one
Ladder [SAT'04]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Bitwise [SAT'07]	$\mathcal{O}(n\ log_2\ n)$	$\mathcal{O}(log_2\ n)$	at-most-one
Commander [CFV'07]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Product [ModRef'10]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Sequential [CP'05]	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$	at-most-k
Totalizer [CP'03]	$\mathcal{O}(nk)$	$\mathcal{O}(n\ log_2\ n)$	at-most-k
Sorters [JSAT'06]	$\mathcal{O}(n\ log_2^2\ n)$	$\mathcal{O}(n\ log_2^2\ n)$	at-most-k

Encodings for cardinality constraints

Encoding	Clauses	Variables	Type
Pairwise	$\mathcal{O}(n^2)$	0	at-most-one
Ladder [SAT'04]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Bitwise [SAT'07]	$\mathcal{O}(n\ log_2\ n)$	$\mathcal{O}(log_2\ n)$	at-most-one
Commander [CFV'07]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Product [ModRef'10]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	at-most-one
Sequential [CP'05]	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$	at-most-k
Totalizer [CP'03]	$\mathcal{O}(nk)$	$\mathcal{O}(n\ log_2\ n)$	at-most-k
Sorters [JSAT'06]	$\mathcal{O}(n\ log_2^2\ n)$	$\mathcal{O}(n\ log_2^2\ n)$	at-most-k

- Many more encodings exist [PBLib'15]
- They can also be generalized to pseudo-Boolean constraints:
  - $a_1x_1 + a_2x_2 + \ldots + a_nx_n \leq k$

Encodings for cardinality constraints

Properties of cardinality encodings:

- Efficient encodings are arc consistent:
  - $x_1 + x_2 + x_3 + \ldots + x_n \leq k$
  - If more than k variables are assigned 1:
    - unit propagation detects a conflict !
  - If k variables are assigned 1:
    - unit propagation assigns 0 to the remaining variables !
- Cardinality encodings are optimal w.r.t unit propagation
  - For any partial assignment, if that partial assignment is unfeasible then unit propagation will detect a conflict
  - No search is needed !

Encodings for cardinality constraints

Properties of cardinality encodings:

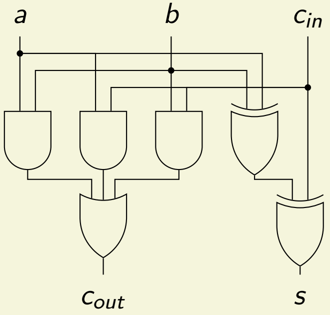
- Do non-optimal cardinality encodings exist?
  - Yes !
  - They can be smaller than optimal cardinality encodings
  - But, their performance can be $10\times$ slower than optimal encodings

- Cardinality encodings must be optimal for performance reasons
- All new cardinality encodings are arc-consistent !
- Efficient encodings for cardinality constraints have a large impact:
  - Better encodings for problems with linear constraints
  - Improving the performance of Boolean optimization solvers
  - ...

Can optimality be extended to other encodings?

[Stronger, Better, Faster: Optimally Propagating SAT Encodings, CADE'15]

Full-adder:

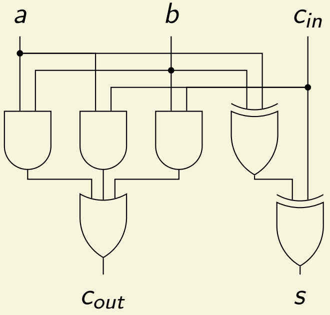


Truth table:

a	b	c_{in}	c_{out}	s
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0

Can optimality be extended to other encodings?

Full-adder:

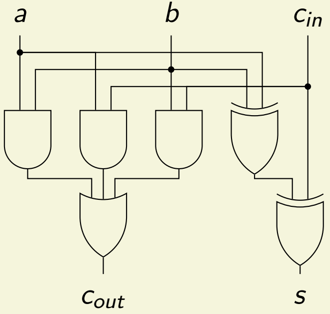


Encoding:

$\{\neg a, \neg b, c_{in}, \neg s\}$	$\{\neg a, b, \neg c_{in}, \neg s\}$
$\{a, \neg b, \neg c_{in}, \neg s\}$	$\{a, b, c_{in}, \neg s\}$
$\{\neg a, \neg b, \neg c_{in}, s\}$	$\{\neg a, b, c_{in}, s\}$
$\{a, b, \neg c_{in}, s\}$	$\{a, \neg b, c_{in}, s\}$
$\{\neg a, \neg b, c_{out}\}$	$\{\neg a, \neg c_{in}, c_{out}\}$
$\{\neg b, \neg c_{in}, c_{out}\}$	$\{a, b, \neg c_{out}\}$
$\{a, c_{in}, \neg c_{out}\}$	$\{b, c_{in}, \neg c_{out}\}$

Can optimality be extended to other encodings?

Full-adder:



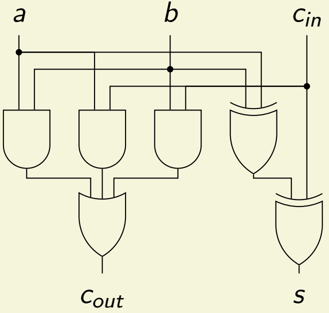
Encoding:

$\{\neg a, \neg b, c_{in}, \neg s\}$	$\{\neg a, b, \neg c_{in}, \neg s\}$
$\{a, \neg b, \neg c_{in}, \neg s\}$	$\{a, b, c_{in}, \neg s\}$
$\{\neg a, \neg b, \neg c_{in}, s\}$	$\{\neg a, b, c_{in}, s\}$
$\{a, b, \neg c_{in}, s\}$	$\{a, \neg b, c_{in}, s\}$
$\{\neg a, \neg b, c_{out}\}$	$\{\neg a, \neg c_{in}, c_{out}\}$
$\{\neg b, \neg c_{in}, c_{out}\}$	$\{a, b, \neg c_{out}\}$
$\{a, c_{in}, \neg c_{out}\}$	$\{b, c_{in}, \neg c_{out}\}$

Is this an optimal encoding?

Can optimality be extended to other encodings?

Full-adder:



Encoding:

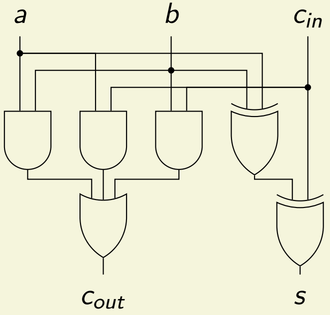
$\{\neg a, \neg b, c_{in}, \neg s\}$ $\{\neg a, b, \neg c_{in}, \neg s\}$
$\{a, \neg b, \neg c_{in}, \neg s\}$ $\{a, b, c_{in}, \neg s\}$
$\{\neg a, \neg b, \neg c_{in}, s\}$ $\{\neg a, b, c_{in}, s\}$
$\{a, b, \neg c_{in}, s\}$ $\{a, \neg b, c_{in}, s\}$
$\{\neg a, \neg b, c_{out}\}$ $\{\neg a, \neg c_{in}, c_{out}\}$
$\{\neg b, \neg c_{in}, c_{out}\}$ $\{a, b, \neg c_{out}\}$
$\{a, c_{in}, \neg c_{out}\}$ $\{b, c_{in}, \neg c_{out}\}$

Is this an optimal encoding?

- No ! Unit propagation does not have the same power as search !
- $UP(c_{cout}, s) = \top$ (no conflict)

Can optimality be extended to other encodings?

Full-adder:



Encoding:

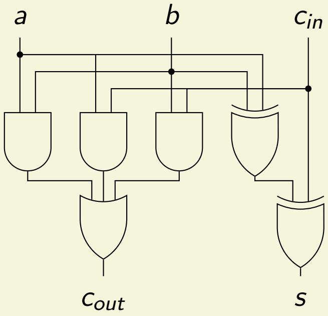
$\{\neg a, \neg b, c_{in}, \neg s\}$	$\{\neg a, b, \neg c_{in}, \neg s\}$
$\{a, \neg b, \neg c_{in}, \neg s\}$	$\{a, b, c_{in}, \neg s\}$
$\{\neg a, \neg b, \neg c_{in}, s\}$	$\{\neg a, b, c_{in}, s\}$
$\{a, b, \neg c_{in}, s\}$	$\{a, \neg b, c_{in}, s\}$
$\{\neg a, \neg b, c_{out}\}$	$\{\neg a, \neg c_{in}, c_{out}\}$
$\{\neg b, \neg c_{in}, c_{out}\}$	$\{a, b, \neg c_{out}\}$
$\{a, c_{in}, \neg c_{out}\}$	$\{b, c_{in}, \neg c_{out}\}$

Is this an optimal encoding?

- No ! Unit propagation does not have the same power as search !
- $UP(c_{cout}, s) = \top$ (no conflict) but $SAT(c_{cout}, s, \neg a) = \bot$ (conflict)
- Unit propagation did not infer that $c_{cout} \wedge s \implies a$!

Can optimality be extended to other encodings?

Full-adder:



Encoding:

$\{\neg a, \neg b, c_{in}, \neg s\}$ $\{\neg a, b, \neg c_{in}, \neg s\}$
$\{a, \neg b, \neg c_{in}, \neg s\}$ $\{a, b, c_{in}, \neg s\}$
$\{\neg a, \neg b, \neg c_{in}, s\}$ $\{\neg a, b, c_{in}, s\}$
$\{a, b, \neg c_{in}, s\}$ $\{a, \neg b, c_{in}, s\}$
$\{\neg a, \neg b, c_{out}\}$ $\{\neg a, \neg c_{in}, c_{out}\}$
$\{\neg b, \neg c_{in}, c_{out}\}$ $\{a, b, \neg c_{out}\}$
$\{a, c_{in}, \neg c_{out}\}$ $\{b, c_{in}, \neg c_{out}\}$

Is this an optimal encoding?

- No ! Unit propagation does not have the same power as search !
- Can we automatically generate optimal encodings?

Finding optimal encodings

Input: $\langle \Sigma, E_0, E_{Ref} \rangle$

```
E ← E_0
PQ.push(λv.?)
while not PQ.empty() do
    core ← PQ.pop()
    foreach v ∈ {x|x ∈ Σ and UP(E)(core)(v) =?} do
        foreach l ∈ {v, ¬v} do
            core′ ← core ⊓ assign(l)
            if SATSolver(E_Ref, core′) = sat then
                PQ.push(core′)
            else
                E ← E ∪{MUS(core′)}
                PQ.compact()
return E
```

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
					$(\neg a \vee c)$	
					$(\neg b \vee c)$	
					$(\neg c \vee d)$	

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	
					$(\neg b \vee c)$	
					$(\neg c \vee d)$	

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	
a	$\neg c$	$\top$	$\bot$	$(\neg a \vee c)$	$(\neg b \vee c)$	
					$(\neg c \vee d)$	

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	$(\neg a \vee c)$
a	$\neg c$	$\top$	$\bot$	$(\neg a \vee c)$	$(\neg b \vee c)$	
					$(\neg c \vee d)$	

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	$(\neg a \vee c)$
a	$\neg c$	$\top$	$\bot$	$(\neg a \vee c)$	$(\neg b \vee c)$	
a, c	b	$\top$	$\top$	–	$(\neg c \vee d)$	

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	$(\neg a \vee c)$
a	$\neg c$	$\top$	$\bot$	$(\neg a \vee c)$	$(\neg b \vee c)$	
a, c	b	$\top$	$\top$	–	$(\neg c \vee d)$	
a, c	$\neg d$	$\top$	$\bot$	$(\neg a \vee d)$		

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	$(\neg a \vee c)$
a	$\neg c$	$\top$	$\bot$	$(\neg a \vee c)$	$(\neg b \vee c)$	$(\neg a \vee d)$
a, c	b	$\top$	$\top$	–	$(\neg c \vee d)$	
a, c	$\neg d$	$\top$	$\bot$	$(\neg a \vee d)$		

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	$(\neg a \vee c)$
a	$\neg c$	$\top$	$\bot$	$(\neg a \vee c)$	$(\neg b \vee c)$	$(\neg a \vee d)$
a, c	b	$\top$	$\top$	–	$(\neg c \vee d)$	
a, c	$\neg d$	$\top$	$\bot$	$(\neg a \vee d)$		
$\neg c, \neg a$	b	$\top$	$\bot$	$(\neg b \vee c)$		

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	$(\neg a \vee c)$
a	$\neg c$	$\top$	$\bot$	$(\neg a \vee c)$	$(\neg b \vee c)$	$(\neg a \vee d)$
a, c	b	$\top$	$\top$	–	$(\neg c \vee d)$	$(\neg b \vee c)$
a, c	$\neg d$	$\top$	$\bot$	$(\neg a \vee d)$		
$\neg c, \neg a$	b	$\top$	$\bot$	$(\neg b \vee c)$		

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	$(\neg a \vee c)$
a	$\neg c$	$\top$	$\bot$	$(\neg a \vee c)$	$(\neg b \vee c)$	$(\neg a \vee d)$
a, c	b	$\top$	$\top$	–	$(\neg c \vee d)$	$(\neg b \vee c)$
a, c	$\neg d$	$\top$	$\bot$	$(\neg a \vee d)$		
$\neg c, \neg a$	b	$\top$	$\bot$	$(\neg b \vee c)$		
$b, c, \neg a$	$\neg d$	$\top$	$\bot$	$(\neg b \vee d)$		

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	$(\neg a \vee c)$
a	$\neg c$	$\top$	$\bot$	$(\neg a \vee c)$	$(\neg b \vee c)$	$(\neg a \vee d)$
a, c	b	$\top$	$\top$	–	$(\neg c \vee d)$	$(\neg b \vee c)$
a, c	$\neg d$	$\top$	$\bot$	$(\neg a \vee d)$		$(\neg b \vee d)$
$\neg c, \neg a$	b	$\top$	$\bot$	$(\neg b \vee c)$		
$b, c, \neg a$	$\neg d$	$\top$	$\bot$	$(\neg b \vee d)$		

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	$(\neg a \vee c)$
a	$\neg c$	$\top$	$\bot$	$(\neg a \vee c)$	$(\neg b \vee c)$	$(\neg a \vee d)$
a, c	b	$\top$	$\top$	–	$(\neg c \vee d)$	$(\neg b \vee c)$
a, c	$\neg d$	$\top$	$\bot$	$(\neg a \vee d)$		$(\neg b \vee d)$
$\neg c, \neg a$	b	$\top$	$\bot$	$(\neg b \vee c)$		
$b, c, \neg a$	$\neg d$	$\top$	$\bot$	$(\neg b \vee d)$		
$\neg a, \neg b, \neg d$	c	$\top$	$\bot$	$(\neg c \vee d)$		

Finding optimal encodings

Given partial assignment ν, reference encoding E_{ref}, goal encoding E, and unassigned literal p; if $UP_E(\nu) = \top$ and $SAT_{E_{ref}}(\nu \cup \{p\}) = \bot$:

- E is not optimal
- E can be extended

ν	p	UP_E	$SAT_{E_{ref}}$	learned	E_{ref}	E
$\emptyset$	a	$\top$	$\top$	–	$(\neg a \vee c)$	$(\neg a \vee c)$
a	$\neg c$	$\top$	$\bot$	$(\neg a \vee c)$	$(\neg b \vee c)$	$(\neg a \vee d)$
a, c	b	$\top$	$\top$	–	$(\neg c \vee d)$	$(\neg b \vee c)$
a, c	$\neg d$	$\top$	$\bot$	$(\neg a \vee d)$		$(\neg b \vee d)$
$\neg c, \neg a$	b	$\top$	$\bot$	$(\neg b \vee c)$		$(\neg c \vee d)$
$b, c, \neg a$	$\neg d$	$\top$	$\bot$	$(\neg b \vee d)$		
$\neg a, \neg b, \neg d$	c	$\top$	$\bot$	$(\neg c \vee d)$		

Finding set-minimal optimal encodings

$E = \{\{\neg a, c\}, \{\neg a, d\}, \{\neg b, c\}, \{\neg b, d\}, \{\neg c, d\}\}$

Is E a set-minimal optimal encoding?

Finding set-minimal optimal encodings

$E = \{\{\neg a, c\}, \{\neg a, d\}, \{\neg b, c\}, \{\neg b, d\}, \{\neg c, d\}\}$

Is E a set-minimal optimal encoding?

- No ! Some clauses may be removed and E is still optimal !
- $\{\neg a, d\}$ is redundant:
  - $a \implies d$ can be inferred from $\{\neg a, c\}$ and $\{\neg c, d\}$
  - $\neg d \implies \neg a$ can be inferred from $\{\neg c, d\}$ and $\{\neg a, c\}$
- Can we minimize E to a set-minimal optimal encoding?

Finding set-minimal optimal encodings

```
Input: E_Opt
foreach c ∈ E_Opt do
    foreach lit ∈ c do
        p ← UP(E_Opt \ c)(¬(c \ lit))
        if p({lit}) =? then
            go to 1
    E_Opt ← E_Opt \ c
return E_Opt
```

Finding set-minimal optimal encodings

E_{opt}	redundant	reason
$\omega_1 = (\neg a \vee c)$	✗	$E_{opt} \setminus \omega_1 \cup \{a\} \not\Longrightarrow c$
$\omega_2 = (\neg a \vee d)$	✓	$E_{opt} \setminus \omega_2 \cup \{a\} \xRightarrow{\omega_1} c \xRightarrow{\omega_5} d$
		$E_{opt} \setminus \omega_2 \cup \{\neg d\} \xRightarrow{\omega_5} \neg c \xRightarrow{\omega_1} \neg a$
$\omega_3 = (\neg b \vee c)$	✗	$E_{opt} \setminus \omega_3 \cup \{b\} \not\Longrightarrow c$
$\omega_4 = (\neg b \vee d)$	✓	$E_{opt} \setminus \omega_4 \cup \{b\} \xRightarrow{\omega_3} c \xRightarrow{\omega_5} d$
		$E_{opt} \setminus \omega_4 \cup \{\neg d\} \xRightarrow{\omega_5} \neg c \xRightarrow{\omega_3} \neg b$
$\omega_5 = (\neg c \vee d)$	✗	$E_{opt} \setminus \omega_5 \cup \{\neg d\} \not\Longrightarrow \neg c$

Generating optimal encodings

- prim: small primitive encodings
  - comparison: lt, slt
  - addition: adder
  - multiplication: mult2
- comp: composition of primitive encodings

			Original enc.		Optimal enc.			
Benchmark	Type	Optimal	#Vars	#Cls	#Vars	#Cls	#minCls	time (s)
lt	prim	✗	10	19	6	18	17	<0.01
slt	prim	✗	8	13	4	6	6	<0.01
adder	prim	✗	9	17	5	14	14	<0.01
mult2	prim	✗	77	182	8	26	21	<0.01
lt-6bit	comp	✗	26	60	13	158	21	24.13
mult-4bit	comp	✗	285	800	16	5322	4942	297.47
plus-3bit	comp	✗	19	39	9	96	96	0.08
plus-aux-3bit	comp	✗	19	39	19	62	42	3.03
plus-4bit	comp	✗	27	58	21	336	336	2.83
plus-aux-4bit	comp	✗	27	58	27	91	65	242.81

Experimental Results

- CVC4 SMT solver [Barret et al. CAV'11]
- 31066 quantifier-free bit-vector benchmarks from SMT-LIB v2.0
  - focus on industrial from industrial applications
- Experiments run on StarExec:
  - timeout: 900 seconds
  - memory limit: 100GB

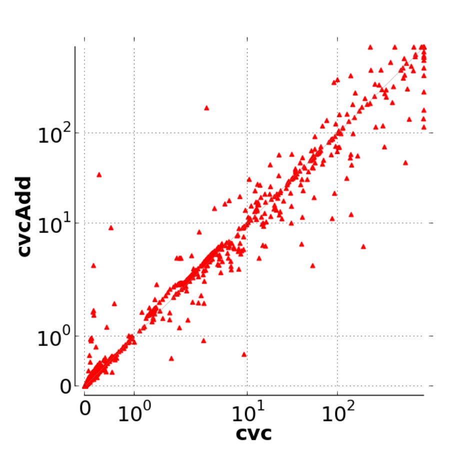
Experimental Results

- CVC4 SMT solver [Barret et al. CAV'11]
- 31066 quantifier-free bit-vector benchmarks from SMT-LIB v2.0
  - focus on industrial from industrial applications
- Experiments run on StarExec:
  - timeout: 900 seconds
  - memory limit: 100GB
- Do optimal encodings improve the performance of SMT solvers?
  - Comparison: cvcLt
  - Addition: cvcAdd
  - Multiplication: cvcMBl2Opt

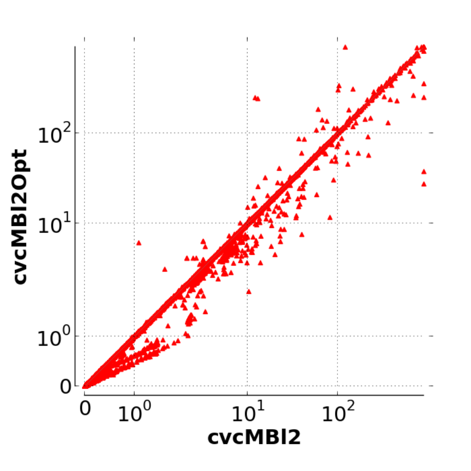
Encoding Comparison

	cvc		cvcLt		cvcAdd		cvcLtAdd		cvcMBl2		cvcMBl2Opt	
set	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)	solved	time (s)
VS3 (11)	**2**	**730.5**	2	900.09	1	496.01	1	48.79	1	120.73	0	0.0
bmc-bv (135)	**135**	**653.4**	134	489.52	135	664.83	134	489.56	135	722.76	135	663.66
bru (52)	39	2619.36	39	2515.33	39	2095.94	**39**	**1945.1**	39	2626.0	39	2639.87
bru2 (65)	56	3367.28	56	3929.27	**56**	**3319.4**	56	3926.21	35	1918.09	36	1087.26
bru3 (79)	40	2791.84	**44**	**5388.8**	39	3497.52	43	5060.69	39	3249.56	40	3332.25
sp (64)	38	2768.64	38	2770.04	40	3104.32	**40**	**3094.8**	38	2738.17	38	2755.44
caly (23)	9	2.13	9	4.34	11	1339.1	**11**	**471.9**	9	16.34	9	5.33
fft (23)	**8**	**874.8**	7	71.94	7	298.1	7	179.53	8	876.93	8	881.75
float (213)	162	11433.73	160	12271.55	**169**	**11504.6**	166	10736.02	159	10214.84	161	10114.83
logs (208)	74	24486.26	75	24956.34	77	26014.95	**79**	**27421.8**	74	24595.51	73	23768.43
mcm (186)	78	7350.21	81	8554.8	**83**	**8996.2**	82	8644.39	78	7364.1	78	7337.21
rubik (7)	6	604.27	**7**	**1378.3**	6	625.01	7	1402.57	6	605.86	6	618.74
spear (1695)	**1690**	**25972.3**	1690	27633.65	1689	26231.45	1690	26133.82	1690	26258.91	1690	26237.04
taca (5)	5	1246.76	5	1075.59	**5**	**957.8**	5	1107.8	5	1242.27	5	1266.86
uclid (416)	**416**	**1343.2**	416	1561.67	416	1515.23	416	1705.26	416	1931.34	416	1592.56
uum (8)	2	10.3	2	10.18	2	10.21	**2**	**10.2**	2	10.23	2	10.18
wien (18)	14	14.14	**14**	**14.0**	14	19.39	14	19.45	14	20.93	14	21.63
	2774	86269.1	2779	93525.46	2789	90690.02	**2792**	**92397.95**	2748	84512.58	2750	82333.04

Optimal vs. Non-Optimal: Adder Encoding



Optimal vs. Non-Optimal: Multiplier Encoding



Conclusions

- Optimal encodings exist for any Boolean formula !
- Computing optimal encodings is exponential, but:
  - Feasible for small encodings
  - Small encodings can be composed into larger encodings:
    - Composition is optimal for addition and comparison
    - Composition is not optimal for multiplication
- Optimal encodings outperform non-optimal encodings !

Conclusions

- Optimal encodings exist for any Boolean formula !
- Computing optimal encodings is exponential, but:
  - Feasible for small encodings
  - Small encodings can be composed into larger encodings:
    - Composition is optimal for addition and comparison
    - Composition is not optimal for multiplication
- Optimal encodings outperform non-optimal encodings !

- Ongoing work:
  - Formalization of optimal encodings [CADE'15]
  - Improved generation of optimal encodings with auxiliary variables
  - Measure how far an encoding is from an optimal encoding:
    - Predict the performance of different encodings

References

Tower of Hanoi Encodings:

H. Kautz and B. Selman. Planning as Satisfiability. ECAI 1992: 359-363

S. Prestwich. Variable Dependency in Local Search: Prevention Is Better Than Cure. SAT 2007: 107-120

R. Martins and I. Lynce. Effective CNF Encodings of the Towers of Hanoi. LPAR 2008.

Cardinality and Pseudo-Boolean Encodings:

C. Ansotegui and F. Manyá. Mapping problems with finite-domain variables into problems with boolean variables. SAT 2004: 1-15 (Ladder)

S. Prestwich. Variable Dependency in Local Search: Prevention Is Better Than Cure. SAT 2007: 107-120 (Bitwise)

W. Klieber and G. Kwon. Efficient CNF Encoding for Selecting 1 from N Objects. CFV 2007 (Commander)

J. Chen. A New SAT Encoding of the At-Most-One Constraint. MofRef 2010 (Product)

C. Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. CP 2005: 827-831 (Sequential)

References

Cardinality and Pseudo-Boolean Encodings:

O. Bailleux and Y. Boufkhad. Efficient CNF Encoding of Boolean Cardinality Constraints. CP 2003: 108-122 (Totalizer)

N. Een and N. Sörensson. Translating pseudo-Boolean Constraints into SAT. JSAT 2006 (2): 1-26 (Sorters)

Peter Steinke. A C++ Toolkit for Encoding Pseudo-Boolean Constraints into CNF. http://tools.computational-logic.org/content/pblib.php

CVC4 SMT solver:

C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, C. Tinelli. CVC4. CAV 2011: 171-177

Optimal Encodings:

M. Brain, L. Hadarean, D. Kroening, and R. Martins. Stronger, Better, Faster: Optimally Propagating SAT Encodings. CADE 2015 (Submitted)