Towards a Compact and Efficient SAT-Encoding of Finite Linear CSP

Tomoya Tanjo, Naoyuki Tamura, Mutsunori Banbara

Kobe University, Japan

ModRef 2010, 6th September 2010

Background

Recently, SAT-based approaches become applicable for solving hard and practical problems.

A SAT-based CSP solver Sugar became a winner of GLOBAL categories of the 2008 and 2009 International CSP Solver Competitions.

- The order encoding used in Sugar shows a good performance for a wide variety of problems.
  - Open Shop Scheduling [Tamura *et al.*, CP2006]
  - Job Shop Scheduling [Koshimura *et al.*, 2010]
  - Test Case Generation [Banbara *et al.*, LPAR2010]
  - Two-Dimensional Strip Packing [Soh *et al.*, RCRA2008]

Overview of Order Encoding

A propositional variable $P(x \le a)$ is introduced for each integer variable x and its domain value a where $P(x \le a)$ is defined as true iff $x \le a$.

Advantage

- It is more efficient than others such as the log encoding.
- Because the *Bounds Propagation* of CSP solvers can be achieved by the *Unit Propagation* of SAT solvers.

Overview of Order Encoding

A propositional variable $P(x \le a)$ is introduced for each integer variable x and its domain value a where $P(x \le a)$ is defined as true iff $x \le a$.

Advantage

- It is more efficient than others such as the log encoding.
- Because the *Bounds Propagation* of CSP solvers can be achieved by the *Unit Propagation* of SAT solvers.

Drawback

- It generates too large SAT instances when the domain size of original CSP is large.
- Because each ternary constraint is encoded into $O(d^2)$ clauses where d is the maximum domain size of integer variables while the log encoding requires $O(\log d)$ clauses.

Proposal of Compact Order Encoding

Proposal of Compact Order Encoding

In this talk, we propose a new SAT encoding method that is compact and efficient.

Proposal of Compact Order Encoding

Proposal of Compact Order Encoding

In this talk, we propose a new SAT encoding method that is compact and efficient.

Compact Order Encoding (C.O.E.)

- Each integer variable is represented by a numeric system of base $B \geq 2$.
- Each digit is encoded by using the order encoding.
- It is an integration and generalization of the order and log encodings.
    - C.O.E. with $B \geq d$ is equivalent to the order encoding.
    - C.O.E. with $B = 2$ is equivalent to the log encoding.

Summary of Compact Order Encoding

	Order Encoding ($B \geq d$)	Compact Order Encoding	Log Encoding ($B = 2$)
Representation of integers	Unary	Base B	Binary
Size of SAT instance	Large	⟵⟶	Small
#clauses	$O(d^2)$	$O(B^2 \log_B d)$	$O(\log d)$
Propagation	Fast	⟵⟶	Slow
#carry ripples	0	$O(\log_B d)$	$O(\log d)$

- Scalability
  - It requires $O(B^2 \log_B d)$ clauses for each ternary constraint.
- Efficiency
  - It enables the *Bounds Propagation* in the most significant digit.
  - It requires $O(\log_B d)$ carry ripples.

Summary of Compact Order Encoding

	Order Encoding ($B \geq d$)	Compact Order Encoding ($B = \lceil \sqrt{d} \rceil$)	Log Encoding ($B = 2$)
Representation of integers	Unary	Base $\lceil \sqrt{d} \rceil$	Binary
Size of SAT instance	Large	←→	Small
#clauses	$O(d^2)$	$O(d)$	$O(\log d)$
Propagation	Fast	←→	Slow
#carry ripples	0	1	$O(\log d)$

- Scalability
  - It requires $O(d)$ clauses for each ternary constraint.
- Efficiency
  - It enables the *Bounds Propagation* in the most significant digit.
  - It requires only one carry ripple.

Summary of experimental results

To confirm the effectiveness of C.O.E., we used the following benchmarks.

Sequence Problem of length n

- It is the handmade problem to evaluate the basic performance of C.O.E. for various bases.
- Only C.O.E. with $B = \lceil \sqrt{d} \rceil$ solved all 5 instances within 2 hours while the order encoding ($B \geq d$) and the log encoding ($B = 2$) solved 2 instances.

Open Shop Scheduling Problem (OSSP)

- We evaluate the performance for a practical application.
- C.O.E. with $B = \lceil \sqrt{d} \rceil$ is compared with other encodings and the state-of-the-art CSP solvers, choco 2.11 and Mistral 1.550.
- Among them, C.O.E. showed the best performance.

Summary of experimental results

To confirm the effectiveness of C.O.E., we used the following benchmarks.

Sequence Problem of length n

- It is the handmade problem to evaluate the basic performance of C.O.E. for various bases.
- Only C.O.E. with $B = \lceil \sqrt{d} \rceil$ solved all 5 instances within 2 hours while the order encoding ($B \geq d$) and the log encoding ($B = 2$) solved 2 instances.

Open Shop Scheduling Problem (OSSP)

- We evaluate the performance for a practical application.
- C.O.E. with $B = \lceil \sqrt{d} \rceil$ is compared with other encodings and the state-of-the-art CSP solvers, choco 2.11 and Mistral 1.550.
- Among them, C.O.E. showed the best performance.

Evaluation for efficiency: OSSP benchmark

Benchmark instances

- A benchmark set by Brucker *et al.* is used for evaluation.
- This is the most difficult benchmark set and it includes some instances that were not closed until 2006.
- As OSSP instances, `j6-*` and `j7-*` are chosen (18 instances).
- The makespan is set to the most difficult (unsatisfiable) case.
- Each OSSP instance is translated to XCSP format as used in the CSP Solver Competition.

Evaluation for efficiency: OSSP benchmark

We compared the CPU times (including encoding times) of the following solvers.

- Order Encoding + MiniSat 2.0
- C.O.E. ($B = \lceil \sqrt{d} \rceil$) + MiniSat 2.0
- Log Encoding + MiniSat 2.0
- choco 2.11 (with arguments used in the CSP Solver Competition)
- Mistral 1.550 (with no arguments)

Comparison of CPU times

Instance	Size	Order	C.O.E.	Log	choco	Mistral
j6-per0-0	6x6	127.80	22.27	384.42	975.85	110.47
j6-per0-1	6x6	3.56	3.23	3.88	33.86	0.00
j6-per0-2	6x6	4.97	3.67	6.30	54.88	0.15
j6-per10-0	6x6	5.37	3.58	6.06	27.44	0.40
j6-per10-1	6x6	3.62	3.13	3.57	12.14	0.01
j6-per10-2	6x6	4.06	3.28	4.65	98.65	0.14
j6-per20-0	6x6	3.56	3.46	4.04	0.42	0.01
j6-per20-1	6x6	3.54	3.28	3.51	0.43	0.01
j6-per20-2	6x6	3.93	3.34	3.81	0.44	0.01
j7-per0-0	7x7	T.O.	T.O.	T.O.	T.O.	T.O.
j7-per0-1	7x7	56.16	11.18	119.52	T.O.	27.10
j7-per0-2	7x7	36.15	8.35	85.39	T.O.	49.92
j7-per10-0	7x7	56.01	15.47	100.07	T.O.	76.81
j7-per10-1	7x7	24.98	7.74	66.32	0.53	0.97
j7-per10-2	7x7	497.15	298.91	2804.06	T.O.	546.06
j7-per20-0	7x7	4.43	4.17	5.18	0.54	0.12
j7-per20-1	7x7	13.38	5.54	19.80	T.O.	16.82
j7-per20-2	7x7	24.38	7.91	32.37	T.O.	26.76
	#solved	17	17	17	11	17
	Average	51.36	24.03	214.88	80.53	50.34

Evaluation for scalability: OSSP benchmark

Benchmark instances

- To evaluate the scalability, we also use the instances generated by multiplying the process times by some constant factor c.
- The factor c is varied within 1, 10, 50, 100, 200, and 1000.

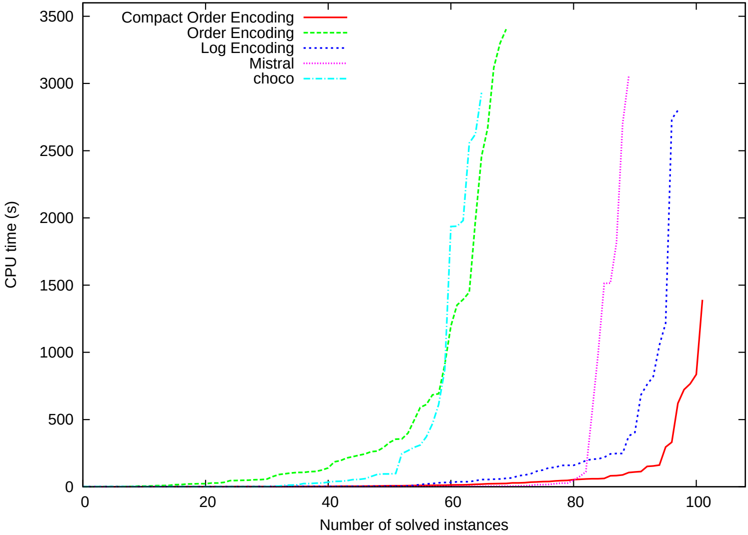
- We compared the number of solved instances of the following solvers.
  - Order Encoding + MiniSat 2.0
  - C.O.E. ($B = \lceil\sqrt{d}\rceil$) + MiniSat 2.0
  - Log Encoding + MiniSat 2.0
  - choco 2.11 (with arguments used in the CSP Solver Competition)
  - Mistral 1.550 (with no arguments)

Comparison of the number of solved instances

Factor c	Domain size d	Order	C.O.E.	Log	choco	Mistral
1	$d \approx 10^3$	17	17	17	11	17
10	$d \approx 10^4$	16	17	17	10	16
50		15	17	16	11	16
100	$d \approx 10^5$	12	17	16	12	15
200		10	17	16	11	14
1000	$d \approx 10^6$	0	17	16	11	12
	Total	70	102	98	66	90

- C.O.E. solved 102 instances out of 108 instances.
- C.O.E. can handle very large domain size such as $d \approx 10^6$.
- When $c = 1000$, C.O.E. generates about 65 MB SAT instances while the order encoding generates more than 13 GB SAT instances in average.

Cactus plot of 108 instances



Conclusion

- In this talk, we presented a new SAT encoding method named compact order encoding.
- The feature of the compact order encoding is:
  - It is a generalization of the order and log encodings.
  - It is efficient. It is more efficient than the log encoding in general because it requires less carry ripples.
  - It is scalable. Each ternary constraint is encoded to $O(B^2 \log_B d)$ clauses where B is the base and d is the domain size. It is much less than $O(d^2)$ clauses of the order encoding.
- We confirmed these observations through some experimental results.

Generated SAT instances (MB)

Factor c	Order	C.O.E.	Log
1	9.43	1.68	1.24
10	107.77	5.66	1.80
50	594.54	13.55	2.12
100	1212.20	19.37	2.27
200	2499.86	27.64	2.43
1000	13467.21	65.46	2.78

- When $c = 1000$, C.O.E. generates about 65 MB SAT instance while the order encoding generates more than 13 GB SAT instances in average.

Runtime memory consumption (MB)

Factor c	Order	C.O.E	Log
1	40.79	11.89	20.71
10	383.25	25.74	27.17
50	1906.92	45.91	25.97
100	3369.82	62.87	26.15
200	6272.71	87.40	28.25
1000	-	187.57	32.19

- When $c = 200$, C.O.E. uses about 87 MB while the order encoding uses more than 6 GB in average.

Sequence Problem

To evaluate the basic performance of C.O.E., we use the following handmade problem.

Sequence Problem

A sequence problem of length n is defined as follows.

$$x_i \in \{0..n-1\} \quad (0 \le i \le n)$$
$$\bigwedge_{i=0}^{n-1} x_i + 1 \le x_{i+1}$$

- This problem is unsatisfiable for any n since there are $n+1$ variables to be arranged in the range of size n.
- To compare the performance of various bases, $\lceil \sqrt[m]{n} \rceil$ ($m \in \{1, 2, 3, 4\}$) and 2 are chosen as a base B.
- The length n is varied within 5000, 8000, 10000, 20000, and 30000.

Comparison of the CPU times

n	Order ($m=1$)	C.O.E. $m=2$	C.O.E. $m=3$	C.O.E. $m=4$	Log ($B=2$)
5000	14.29	64.78	76.58	103.33	596.80
8000	47.02	189.03	212.21	384.93	2611.44
10000	M.O.	382.95	650.58	526.52	T.O.
20000	M.O.	1527.46	4889.55	6311.37	T.O.
30000	M.O.	4631.40	T.O.	T.O.	T.O.

- Only C.O.E. solved all given instances.
- The order and log encodings could not solve the instance when $n \geq 10000$.
- Choosing $m = 2$ (i.e. $B = \lceil \sqrt{n} \rceil$) is the most effective choice for this problem.

Comparison of generated SAT instances (MB)

n	Order ($m = 1$)	C.O.E. $m = 2$	C.O.E. $m = 3$	C.O.E. $m = 4$	Log ($B = 2$)
5000	1005.64	56.46	28.93	21.36	16.54
8000	2643.70	122.94	51.89	38.37	26.74
10000	4155.76	173.65	72.35	48.11	37.46
20000	17955.93	509.32	201.49	119.19	81.99
30000	40954.37	977.52	352.53	227.37	127.40

- C.O.E. generates much smaller SAT instances even when $m = 2$.
- When $n = 30000$, the size of the order encoding is more than 40 GB.

Runtime memory consumption (MB)

Length n	Order Encoding	C.O.E. $m=2$	C.O.E. $m=3$	C.O.E. $m=4$	Log Encoding
5000	4827.61	231.84	121.57	104.86	200.80
8000	13073.18	435.35	221.14	194.59	502.07
10000	M.O.	622.10	377.71	261.69	T.O.
20000	M.O.	1795.87	1028.27	1035.88	T.O.
30000	M.O.	3220.83	T.O.	T.O.	T.O.

- When $n = 5000$ and 8000, the order encoding proved satisfiability with no decision.
- When $n = 8000$, C.O.E. uses less memory than the log encoding.

Arguments of CSP solvers

We use the command line arguments used in the 2009 International CSP Solver Competition.

- choco
  -randval true -h 1 -ac 32 -saclim 60 -s true -verb 0 -seed 11041979
- Mistral
  No arguments

Comparison of the size of encoded-SAT instance

Let d be the maximum domain size of x, y, z and $B \geq 2$ be a base.

Constraint	Direct	Order	C.O.E	Log
$x \leq a$	$O(d)$	$O(1)$	$O(\log_B d)$	$O(\log_2 d)$
$x \leq y$	$O(d^2)$	$O(d)$	$O(B \log_B d)$	$O(\log_2 d)$
$z = x + a$	$O(d^2)$	$O(d)$	$O(B \log_B d)$	$O(\log_2 d)$
$z = x + y$	$O(d^3)$	$O(d^2)$	$O(B^2 \log_B d)$	$O(\log_2 d)$

- Each ternary constraint can be encoded $O(B^2 \log_B d)$ SAT clauses by using C.O.E. in the worst case.
- It is much less than $O(d^3)$ SAT clauses of the direct encoding.