Optimizing Simple Tabular Reduction with a Bitwise Representation

Ruiwei Wang[1] Wei Xia[2] Roland H. C. Yap[2] Zhanshan Li[1]

[1]School of Software, Jilin University, Changchun, China [2]School of Computing, National University of Singapore, Singapore

wangrw13@mails.jlu.edu.cn, {xiawei, ryap}@comp.nus.edu.sg, lizs@jlu.edu.cn

1 Contributions

- ★ ***Two bitwise representations of table constraint:*** We propose to encode table and compressed (Cartesian product) table constraint with bit vectors.
- ★ ***GAC algorithms STRbit and STRbit-C:*** We give two new GAC algorithms for table constraints on the bitwise representation.
- ★ ***Experiments:*** We investigate the performace of our algorithms and compare them with the state-of-the-art algorithms over CSP competition benchmarks.

2 Background

- ★ ***Generalised arc consistency (GAC):*** A CSP is GAC iff any domain value of a variable has at least one support in any constraint including the variable.
- ★ ***Table constraint***: A positive (negative) table constraint C is a constraint whose relation *rel*(C) is represented in a table of allowed (disallowed) tuples.
- ★ ***Simple tabular reduction (STR):*** A GAC algorithm for table constraint, which shrinks table dynamically during backtrack search.

3 STRbit algorithm

Represent table with bit

Bit table: encode the supports of each literal (a variable-value pair) in a constraint's dual table with bit vectors.

standard table

	x	y	z	
0	a	a	b	p_1
1	a	b	b	
2	a	b	c	
3	a	c	c	
4	b	a	c	p_2
5	b	c	c	
6	c	a	a	
7	c	b	a	

dual table for STR3 algorithm

x		y		z	
a	1,2,3,4 ◁	a	1,5,7 ◁	a	7,8 ◁
b	5,6 ◁	b	2,3,8 ◁	b	1,2 ◁
c	7,8 ◁	c	4,6 ◁	c	3,4,5,6 ◁

◁ indicate the position of the last support

bit table

x		y		z	
a	(p_1,1111) ◁	a	(p_1,1000),(p_2,1010) ◁	a	(p_2,0011) ◁
b	(p_2,1100) ◁	b	(p_1,0110),(p_2,0001) ◁	b	(p_1,1100) ◁
c	(p_2,0011) ◁	c	(p_1,0001),(p_2,0100) ◁	c	(p_1,0011),(p_2,1100) ◁

The table is paritioned into 2 parts, p_1 and p_2, and each part corresponds to one **4-bit vector** (**VAL**), which records the validity of tuples. E.g. if all tuples are valid, VAL(p_1)=(1111) and VAL(p_2)=(1111).

Algorithm

STRbit adapts the STR algorithm and maintains GAC during search by (1) updating the validity of tuples represented by bit vectors, and (2) seeking supports for domain values using bit operations over bit table.

Example

Assume c ∉ dom(Z). Use supp(var,val)[p_i] to refer to the corresponding bit vector in the bit table.

Step (1) (update the validity of tuples with (Z,c)'s supports before)

VAL(p_1) = ¬ (supp(Z,c)[p_1] & VAL(p_1)) & VAL(p_1) = (1100)

VAL(p_2) = ¬ (supp(Z,c)[p_2] & VAL(p_2)) & VAL(p_2) = (0011)

Step (2) (seek supports for domain values using new VAL)

E.g. (X,a) is valid, as supp(X,a)[p_1] & VAL(p_1) = (1100) != (0000).

(Y,c) is invalid, as supp(Y,c)[p_i] & VAL(p_i) is (0000) for both p_1 and p_2.

Complexity analysis

Theorem 1 The accumulated time cost of *r* arity constraint *C* in STRbit along a single path of length *m* in the search tree is $O(L_{bit} + r^2d^2 + m)$.

L_{bit} denotes the number of bit supports in the bit table. *r* is the arity. *d* is the domain size.

4 STRbit-C algorithm

Represent c-table with bit

Bit c-table: encode the c-tuple supports of each literal in a constraint's dual table with bit vectors.

standard c-table (the Cartesion product)

	x	y	z	
0	{a}	{a,b}	{b}	p_1
1	{a}	{b.c}	{c}	
2	{b}	{a,c}	{c}	
3	{c}	{a,b}	{a}	

bit c-table

x		y		z	
a	(p_1,1100) ◁	a	(p_1,1011) ◁	a	(p_1,0001) ◁
b	(p_1,0010) ◁	b	(p_1,1101) ◁	b	(p_1,1000) ◁
c	(p_1,0001) ◁	c	(p_1,0110) ◁	c	(p_1,0110) ◁

◁ indicate the position of the last support

The c-table is paritioned into 1 part p_1, which corresponds to one **4-bit vector** (**VAL**). Assume all c-tuples are valid, then the bit vector is (1111). A c-tuple is valid if any one tuple of it is valid.

Algorithm

Smililar as STRbit, STRbit-C maintains GAC by (1) updating the validity of c-tuples, which is also represented by bit vector, and (2) seeking supports for domain values with bit operations over bit c-table.

Example

Assume a ∉ dom(Y), and dom(Y)={b, c}, Use supp(var,val)[p_i] to refer to the corresponding bit vector in the bit c-table.

Step (1) (update the validity of c-tuples with Y's supports before)

VAL(p_1) = (supp(Y,b)[p_1] | supp(Y,c)[p_1]) & VAL(p_1) = (1111)

Step (2) (seek supports for domain values using new VAL)

E.g. (X, a) is valid, as (1111) & VAL(p_1) = (1100) != (0000).

Or assume a ∉ dom(X), and dom(X)={b, c},

Step (1): VAL(p_1) = (supp(X,b)[p_1] | supp(X,c)[p_1]) & VAL(p_1) = (0011)

Step (2): (Z,b) becomes invalid, as supp(Z,b)[p_1] & (VAL(p_1)) = (0000).

Complexity analysis

Theorem 2 The accumulated time cost of *r* arity constraint *C* in STRbit-C along a single path of length *m* in the search tree is $O(dL_{bitc} + r^2d^2 + m)$.

L_{bitc} denotes the number of bit c-supports included in the bit c-table.

5 Experiments

★ Purpose

Evaluate the performance of STRbit and STRbit-C and compare them with the state-of-the-art GAC algorithms for table constraints.

★ Benchmarks

Both structured and unstructured problem instances from CSP competition. (896 instances in total)

★ Tools

AbsCon: a CP solver with various algorithms and heuristics implemented.

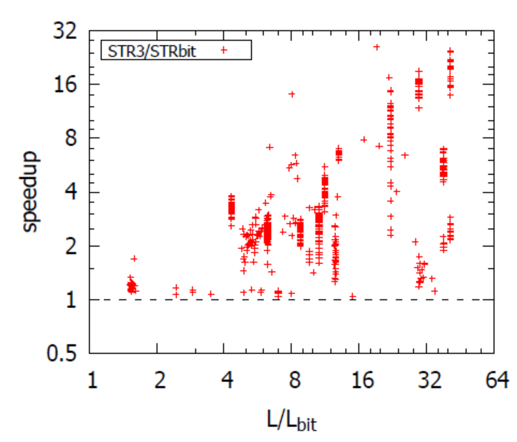
- Variable heuristic: dom/ddeg
- Value heuristic: lexico

★ Runtime distribution for unstructured instances

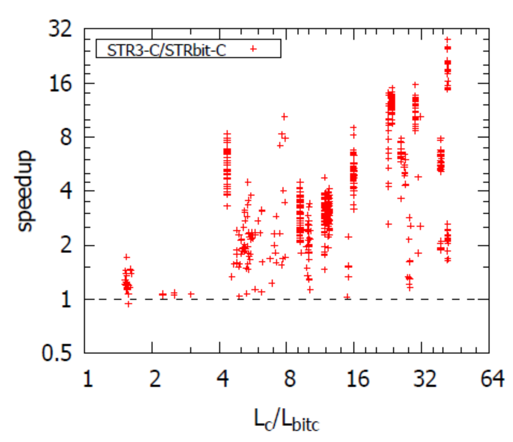
	Instances	#	STRbit	STRbit-C	STR2	STR2-C	STR3	STR3-C	MDDc	L/L_{bit}	L_c/L_{bitc}	L/L_c	L/L_{bitc}	$avgP$	L	L_c
R	rand-3	50	16.74	**12.15**	52.56	31.54	41.12	38.95	29.18	6.25	12.30	2.35	28.92	0.0567	8K	3K
	rand-3-fcd	50	8.50	**6.09**	29.05	15.77	20.93	19.63	14.77	6.25	12.30	2.35	28.92	0.0573	8K	3K
	rand-8	20	8.43	9.18	**8.20**	10.40	93.85	94.52	13.78	22.11	22.76	1.54	35.00	0.0018	624K	406K
	rand-5-8X	26	70.88	19.71	416.36	112.70	467.80	226.85	**19.43**	13.01	23.62	3.35	79.15	0.0075	497K	148K
	rand-5-4X	50	6.85	**5.10**	61.22	33.92	29.38	26.69	8.68	11.28	15.93	2.54	40.41	0.0406	248K	98K
	rand-5-2X	50	**1.97**	2.64	10.02	11.35	5.22	8.02	4.23	10.62	11.80	1.79	21.09	0.0911	124K	69K
	rand-5	50	**2.02**	3.58	21.90	30.43	4.77	10.57	15.36	8.77	9.18	1.34	12.29	0.2379	62K	46K
	rand-10-60	31	**12.71**	22.18	141.81	268.66	40.51	130.40	29 time-out	4.32	4.34	1.00	4.34	0.2637	512K	511K
	dag-rand	25	12.39	18.38	**9.86**	21.82	198.44	217.87	72.69	29.65	29.99	1.04	31.07	0.0014	2M	2M
	half	16	70.46	**36.99**	233.32	98.07	9 time-out	242.75	57.69	22.15	25.90	5.65	146.28	0.0059	277K	49K
	MDD0.7	6	32.08	**11.93**	392.67	44.96	382.38	63.29	20.56	22.11	26.98	12.42	335.06	0.0197	273K	22K
	MDD0.9	9	3.78	**2.22**	53.40	3.61	30.82	3.92	2.80	22.11	28.25	34.53	975.32	0.0675	273K	7K
	bdd-small	35	2.30	**2.23**	3.21	4.68	17.18	18.00	11.27	40.99	41.86	1.25	52.49	0.0385	1M	829K
	bdd-large	35	3.09	**3.07**	8.70	9.41	42.68	41.43	30.75	38.13	39.01	1.24	48.41	0.0421	103K	83K
S	golombR	16	42.04	**30.12**	50.04	152.53	70.93	47.60	31.46	6.62	30.20	5.54	167.46	0.0206	566K	103K
	uk	29	**2.16**	2.65	7.29	13.45	6.64	8.16	45.83	5.59	5.42	1.16	6.34	0.1806	105K	92K
	ogd	29	**0.98**	1.39	9.70	33.96	4.89	8.46	27.26	7.52	7.09	1.25	8.92	0.2249	199K	165K
	lex	40	0.84	**0.82**	1.76	2.68	1.68	1.84	6.39	4.48	4.44	1.06	4.77	0.1064	12K	12K
	words	40	**2.40**	2.73	6.35	10.03	5.74	6.08	24.76	5.27	5.18	1.11	5.77	0.1181	26K	24K
	lemma	9	10.79	**10.61**	15.46	11.53	30.91	20.89	11.80	9.73	13.51	2.36	31.92	0.077	957	405
	modR	50	7.28	**6.74**	17.58	18.12	17.24	18.04	18.01	12.57	10.06	22.60	227.56	0.0396	12K	542
	cril	1	3.48	**3.07**	4.69	3.21	6.07	3.16	3.65	29.69	14.88	43.70	650.31	0.0563	12K	284
	allInter	11	**15.02**	15.63	16.28	16.52	16.58	16.69	15.98	5.30	5.35	1.18	6.32	0.0603	379	321
	tsp-20	15	1.49	1.69	**1.20**	1.51	1.77	2.14	1.61	1.55	1.55	1.00	1.55	0.002	41K	41K
	tsp-25	15	14.90	16.38	**12.48**	14.71	17.73	19.75	16.28	1.55	1.56	1.00	1.56	0.0021	41K	41K
M	aim-100	17	49.08	50.68	48.78	54.49	48.85	48.13	**43.68**	3.47	2.24	1.55	3.47	0.4967	20	13
	aim-200	8	52.76	57.81	49.74	50.28	53.51	50.87	**42.21**	3.48	2.24	1.55	3.48	0.7299	20	13
	aim-50	24	0.18	0.17	0.18	**0.16**	**0.16**	0.17	**0.16**	3.47	2.23	1.55	3.47	0.4549	20	13
	dubois	5	183.67	181.25	175.90	203.01	173.17	176.09	**167.12**	2.00	2.00	1.00	2.00	0.283	12	12
MC	cc	8	53.85	56.56	61.99	61.73	59.26	53.26	**46.08**	9.14	4.01	2.58	10.35	0.4441	88	34
	ramsey	5	6.24	6.30	7.68	7.77	6.65	**5.66**	5.91	11.29	3.80	2.97	11.29	0.3745	115	38
	jnh	50	0.49	0.51	0.63	**0.47**	0.62	0.51	0.51	30.93	6.04	12.58	76.05	0.2736	787	62
	ii	10	12.51	6.98	1 time-out	6.69	1 time-out	6.73	**6.50**	42.47	10.84	458.13	5K	0.2704	78K	170
	PH-k-j	9	2.69	2.18	39.58	2.31	13.01	**2.01**	2.52	19.81	1.00	84K	84K	0.2363	4M	48

Timeout is 600s. L (L_c) is the mean number of literals in the standard tables (c-tables). L_{bit} (L_{bitc}) is the mean number of bit supports (bit c-supports) in the bit tables (bit c-tables). L/L_{bit}, L_c/L_{bitc}, L/L_c and L/L_{bitc} represent the corresponding compression ratio.

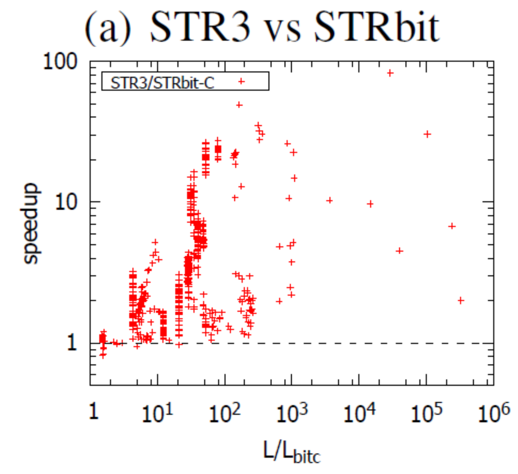
★ Runtime distribution for instances



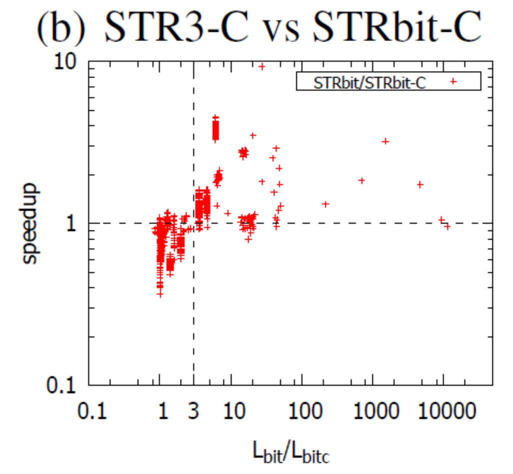
(a) STR3 vs STRbit



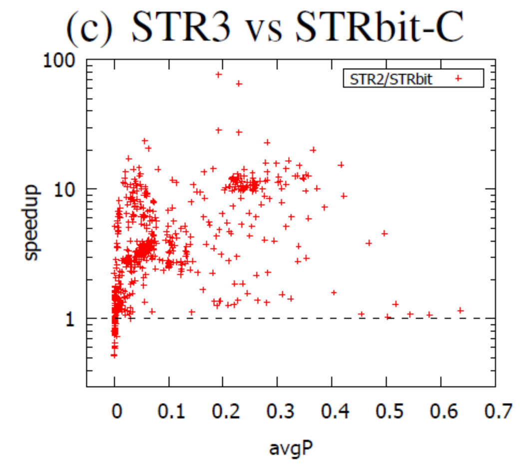
(b) STR3-C vs STRbit-C



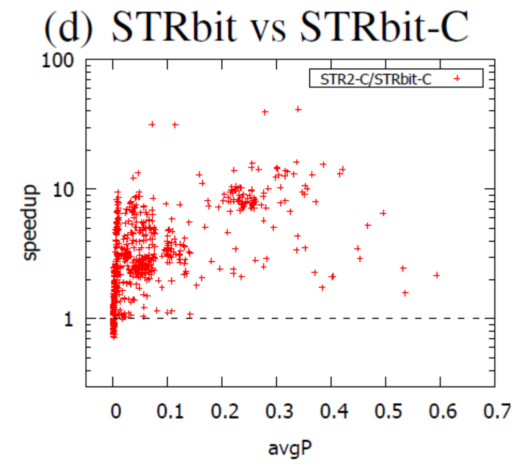
(c) STR3 vs STRbit-C



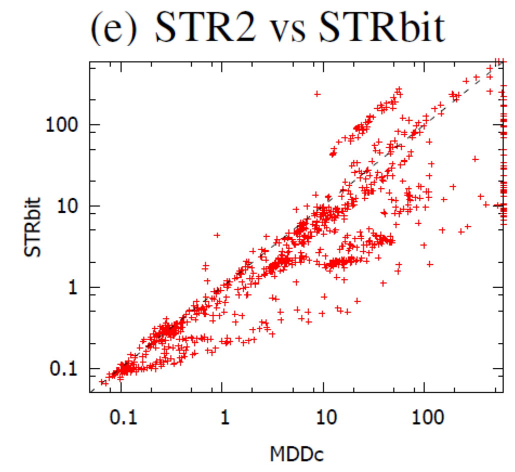
(d) STRbit vs STRbit-C



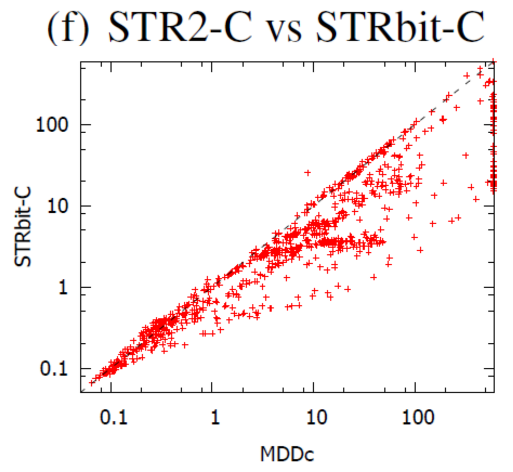
(e) STR2 vs STRbit



(f) STR2-C vs STRbit-C



(g) STRbit vs MDDc



(h) STRbit-C vs MDDc