PW-CT: Extending Compact-Table to Enforce Pairwise Consistency on Table Constraints

Anthony Schneider & Berthe Y. Choueiry

Constraint Systems Laboratory
University of Nebraska-Lincoln

Acknowledgements

- Experiments conducted at UNL's Holland Computing Center
- NSF Grant No. RI-1619344

Outline

- Introduction: Pairwise Consistency definition (fPWC)
- Methods for improving fPWC
  1. Piecewise functionality of relations for quicker filtering
  2. Unique intersections of relations' scopes
  3. Minimal dual graph
  4. Two cases where GAC guarantees fPWC
- Enforcing fPWC with COMPACTTABLE
- Experimental results
- Conclusions

Definitions

- A *subscope* is the set of variables shared by the scopes of two constraints
- A CSP is PWC iff
  - For every tuple t_i in every constraint c_i
  - There is a tuple t_j in every constraint c_j such that

$$\pi_{subscope(c_i,c_j)}(t_i) = \pi_{subscope(c_i,c_j)}(t_j)$$

- A CSP is fPWC iff it is PWC and GAC

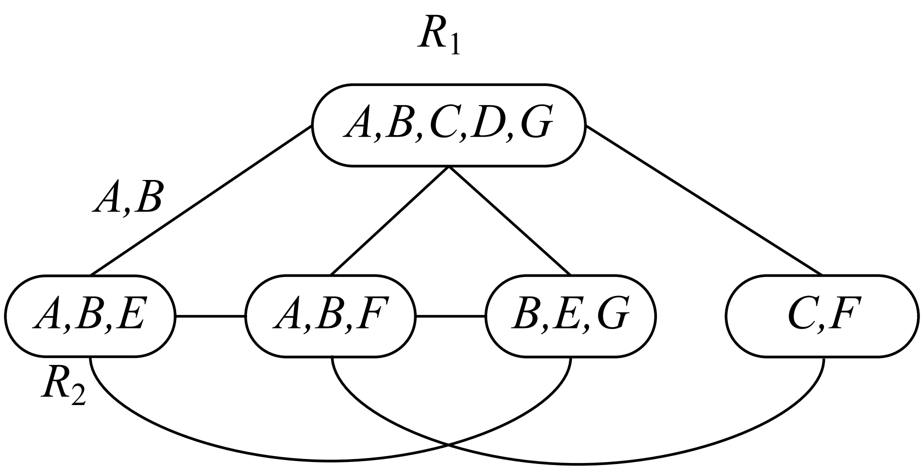
	A	B	E
√	0	0	0
√	0	0	1
√	0	1	0
√	0	1	1
x	1	0	0
x	1	0	1

	A	B	C	D	G
	0	0	0	0	0
	0	0	0	1	0
	0	0	1	0	0
	0	0	1	1	1
	0	1	1	0	1
	0	1	1	1	1
x	1	1	1	1	1

Piecewise Functional Constraints

- Used in PW-AC

[Samaras & Stergiou, JAIR 05]



R_1

	A	B	C	D	G
t_1	0	0	0	0	0
t_2	0	0	0	1	0
t_3	0	0	1	0	0
t_4	0	0	1	1	1
t_5	0	1	1	0	1
t_6	0	1	1	1	1
t_7	1	1	1	1	1

R_2

A	B	E
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1

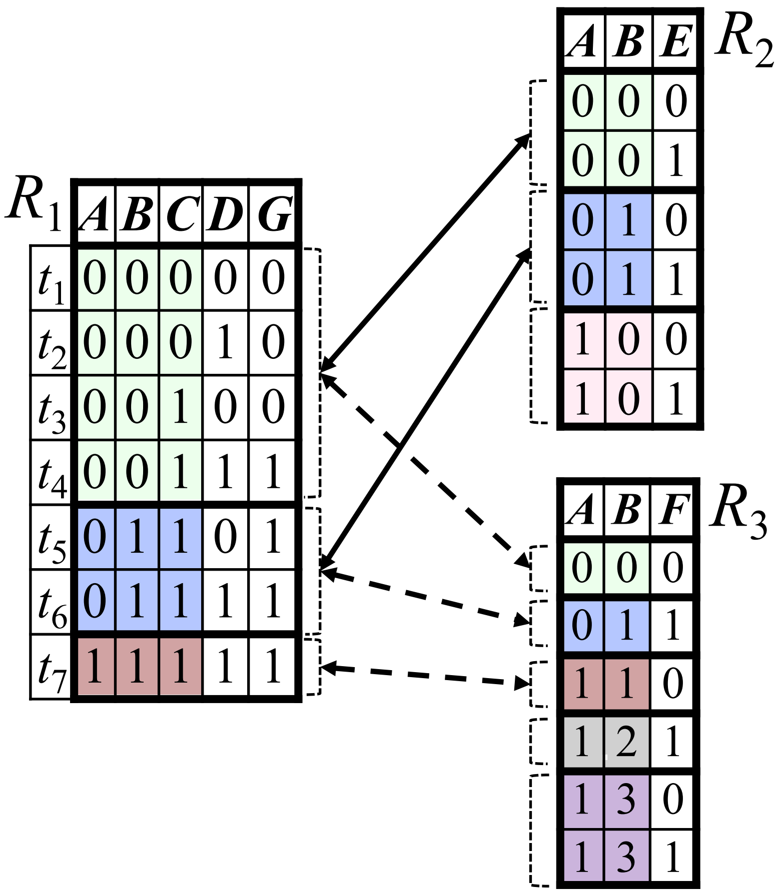
{(A,0),(B,0)}

{(A,0),(B,1)}

X

X

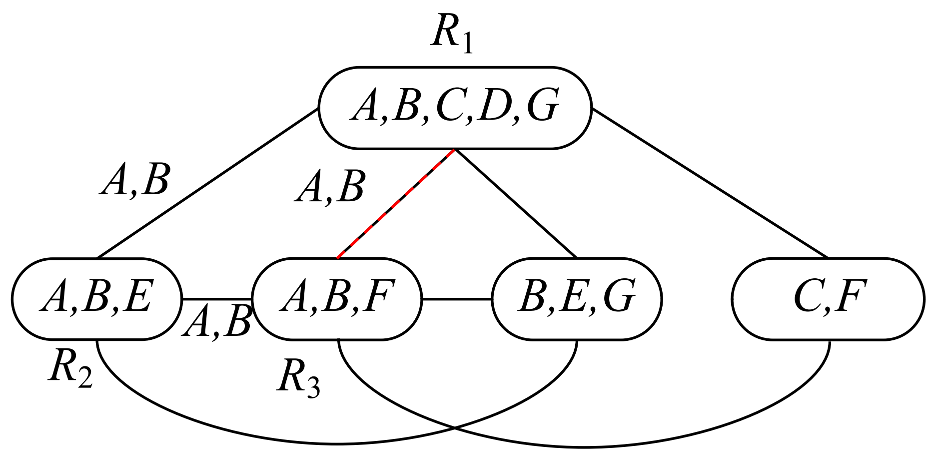
Focus on Subscopes (not relation pairs)



	A	B	C	D	G
t_1	0	0	0	0	0
t_2	0	0	0	1	0
t_3	0	0	1	0	0
t_4	0	0	1	1	1
t_5	0	1	1	0	1
t_6	0	1	1	1	1
t_7	1	1	1	1	1

A	B	E
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1

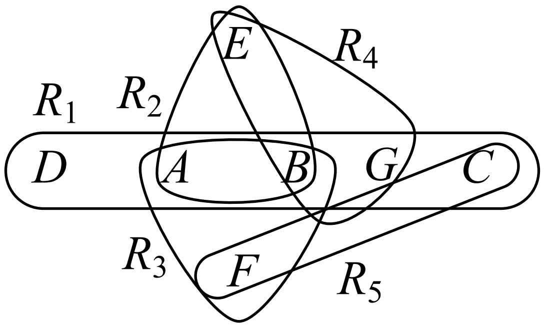
A	B	F
0	0	0
0	1	1
1	1	0
1	2	1
1	3	0
1	3	1



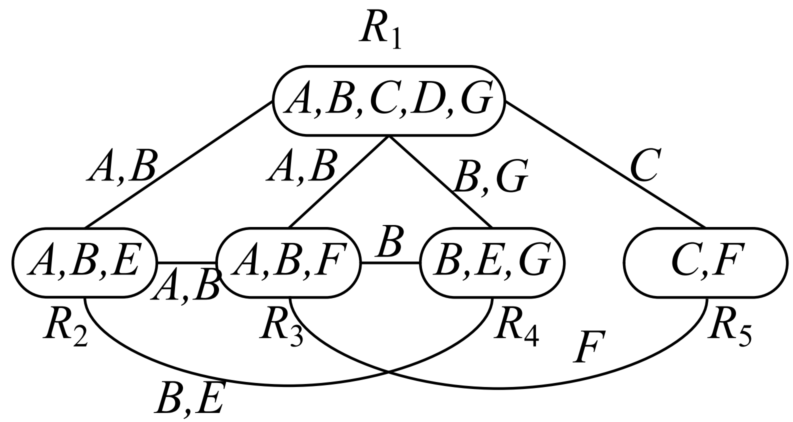
Huge savings in

1. Memory
2. Propagation speed

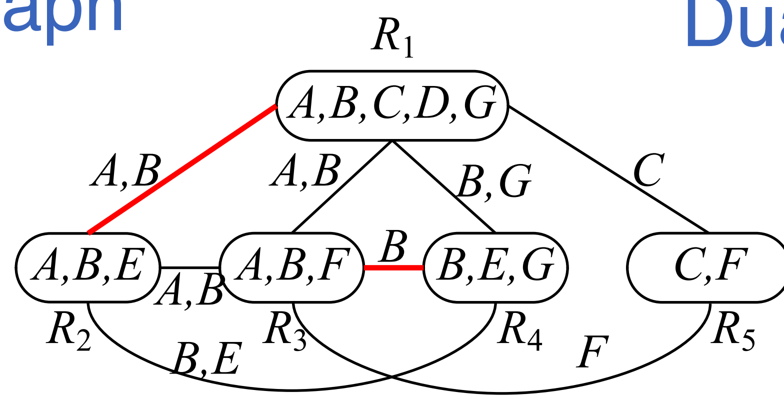
Using the Minimal Dual Graph



Hypergraph



Dual graph



Minimal dual graph [Janssen et al.,1989]

Avg. number of non-trivial subscopes	
Original dual	Minimal dual
399.5	264.3

GAC is fast. Use it.

- Modern GAC algorithms are *fast* and low-memory
  - Compact Table, STRBit
- … and sometimes sufficient for enforcing fPWC
  - Case 1: Subscopes with a single variable [Lecoutre et al., 2016]
  - Case 2: Blocks whose signature contains a variable-value pair removed by a GAC algorithm

	A	B	C	D	G
t_1	0	0	0	0	0
t_2	0	0	0	1	0
t_3	0	0	1	0	0
t_4	0	0	1	1	1
t_5	0	1	1	0	1
t_6	0	1	1	1	1
t_7	1	1	1	1	1

A	B	E
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1

A	B	F
0	0	0
0	1	1
1	1	0
1	2	1
1	3	0
1	3	1

1. GAC removes ⟨*A*,0⟩
2. GAC filters all tuples with ⟨*A*,0⟩
3. Consequently, blocks
   - {⟨*A*,0⟩,⟨*B*,0⟩} and
   - {⟨*A*,0⟩,⟨*B*,1⟩}

   are removed

Bringing It All Together

Challenge: A new algorithm integrating all of these techniques

- Use CT to enforce GAC
- Detect *blocks* of tuples annihilated by CT
  - Dynamically determine blocks at run-time
- After CT runs
  - Check modified constraints for blocks with no living tuples
  - Propagate dead blocks to incident relations via subscopes
- Repeat until fixpoint or failure

Data Structures

Tables

R_1	A	B	C	D
t_1	0	0	1	0
t_2	0	0	0	1
t_3	0	0	1	2
t_4	0	1	2	2
t_5	1	1	1	2
t_6	1	0	2	1
t_7	2	1	0	1
t_8	2	0	2	0

R_2	A	C	F
t_1	0	0	0
t_2	0	1	1
t_3	0	2	1
t_4	1	1	1
t_5	1	2	2
t_6	2	0	2
t_7	2	2	1

R_3	B	C	E
t_1	0	0	0
t_2	0	1	0
t_3	0	1	1
t_4	0	2	1
t_5	1	0	2
t_6	1	1	2
t_7	1	2	2

Living Tuples

R_1	R_2	R_3
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1	1	1
1		

Supports (for R_1)

$A,0$	$A,1$	$A,2$	$B,0$	$B,1$
1	0	0	1	0
1	0	0	1	0
1	0	0	1	0
1	0	0	0	1
0	1	0	0	1
0	1	0	1	0
0	0	1	0	1
0	0	1	1	0

$C,0$	$C,1$	$C,2$	$D,0$	$D,1$	$D,2$
0	1	0	1	0	0
1	0	0	0	1	0
0	1	0	0	0	1
0	0	1	0	0	1
0	1	0	0	0	1
0	0	1	0	1	0
1	0	0	0	1	0
0	0	1	1	0	0

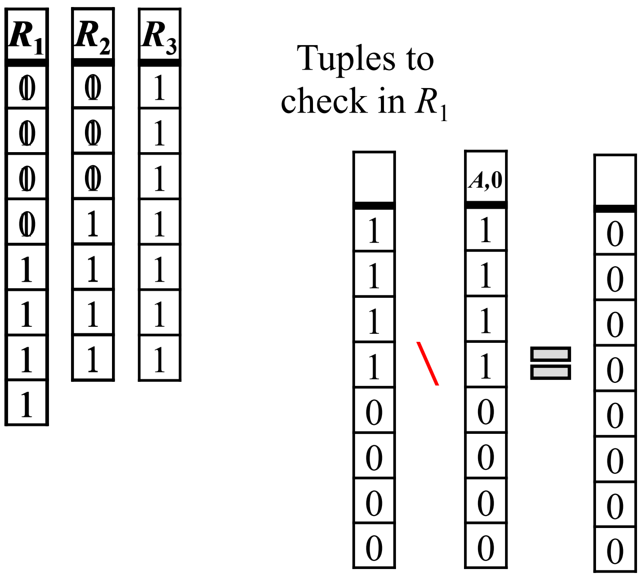
Block $\{\langle A,0\rangle,\langle B,1\rangle\}$ in R_1

&

R_1	$A,0$	$B,1$	=	$Block$
1	1	0		0
1	1	0		0
1	1	0		0
1	1	1		1
1	0	1		0
1	0	0		0
1	0	1		0
1	0	0		0

Brief Example

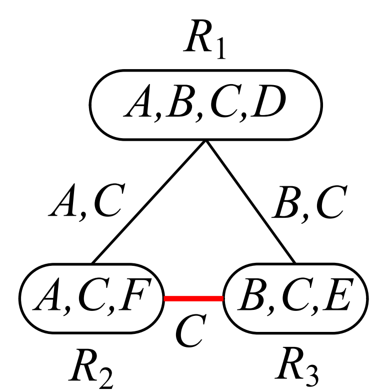
Living Tuples



R_1	A	B	C	D
t_1	0	0	1	0
t_2	0	0	0	1
t_3	0	0	1	2
t_4	0	1	2	2
t_5	1	1	1	2
t_6	1	0	2	1
t_7	2	1	0	1
t_8	2	0	2	0

R_2	A	C	F
t_1	0	0	0
t_2	0	1	1
t_3	0	2	1
t_4	1	1	1
t_5	1	2	2
t_6	2	0	2
t_7	2	2	1

R_3	B	C	E
t_1	0	0	0
t_2	0	1	0
t_3	0	1	1
t_4	0	2	1
t_5	1	0	2
t_6	1	1	2
t_7	1	2	2



Skip blocks for subscope that were filtered by CT

Brief Example

Living Tuples

R_1	R_2	R_3
0	0	1
0	0	1
0	0	1
0	1	1
1	1	1
1	1	1
1	1	1
1		

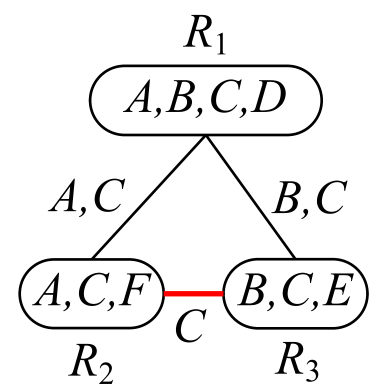
Tuples to check in R_1

		&		
Tuples to check in R_1	$B,0$	$C,1$	=	Block
1	1	1		1
1	1	0		0
1	1	1		1
1	0	0		0
0	0	1		0
0	1	0		0
0	0	0		0
0	1	0		0

R_1	A	B	C	D
t_1	0	0	1	0
t_2	0	0	0	1
t_3	0	0	1	2
t_4	0	1	2	2
t_5	1	1	1	2
t_6	1	0	2	1
t_7	2	1	0	1
t_8	2	0	2	0

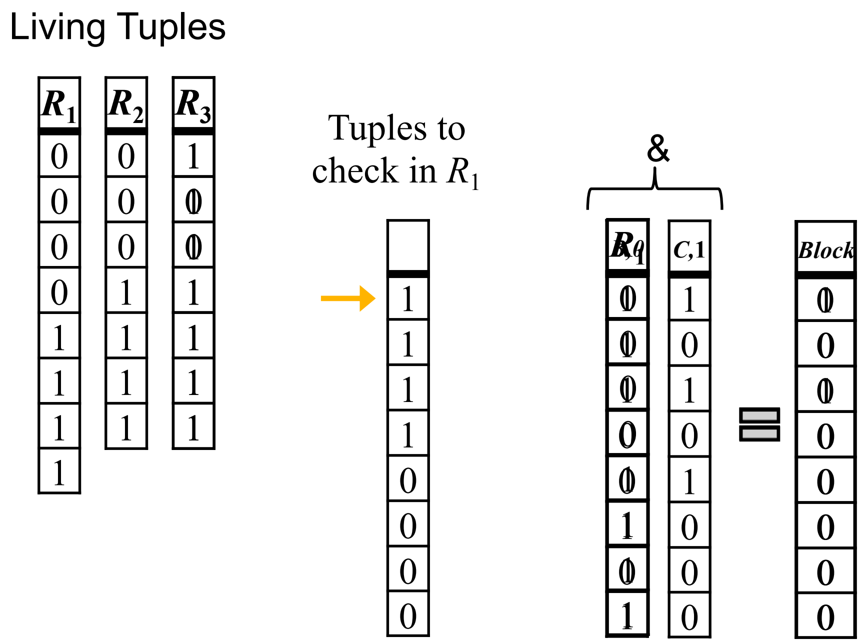
R_2	A	C	F
t_1	0	0	0
t_2	0	1	1
t_3	0	2	1
t_4	1	1	1
t_5	1	2	2
t_6	2	0	2
t_7	2	2	1

R_3	B	C	E
t_1	0	0	0
t_2	0	1	0
t_3	0	1	1
t_4	0	2	1
t_5	1	0	2
t_6	1	1	2
t_7	1	2	2



filtered by CT

Brief Example



R_1

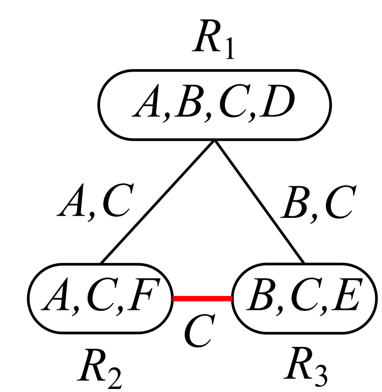
	A	B	C	D
t_1	0	0	1	0
t_2	0	0	0	1
t_3	0	0	1	2
t_4	0	1	2	2
t_5	1	1	1	2
t_6	1	0	2	1
t_7	2	1	0	1
t_8	2	0	2	0

R_2

	A	C	F
t_1	0	0	0
t_2	0	1	1
t_3	0	2	1
t_4	1	1	1
t_5	1	2	2
t_6	2	0	2
t_7	2	2	1

R_3

	B	C	E
t_1	0	0	0
t_2	0	1	0
t_3	0	1	1
t_4	0	2	1
t_5	1	0	2
t_6	1	1	2
t_7	1	2	2



The block was removed by CT – update other relations incident to subscope $\{B,C\}$

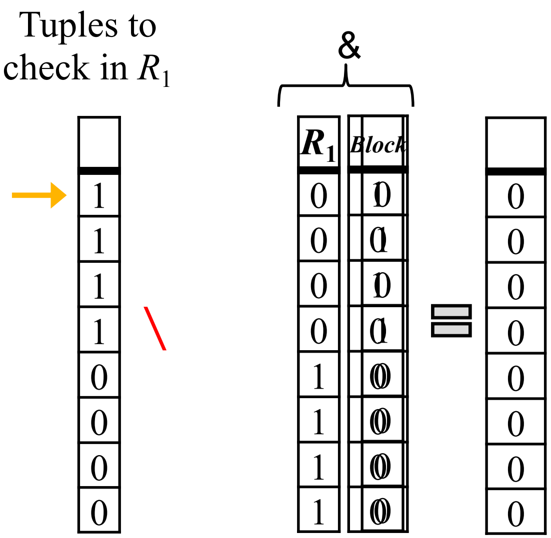
Remove the block from tuples to check

Check if block has any remaining tuples

Brief Example

Living Tuples

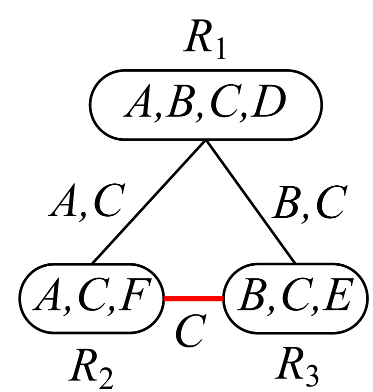
R_1	R_2	R_3
0	0	1
0	0	0
0	0	0
0	1	1
1	1	1
1	1	1
1	1	1
1		



R_1	A	B	C	D
t_1	0	0	1	0
t_2	0	0	0	1
t_3	0	0	1	2
t_4	0	1	2	2
t_5	1	1	1	2
t_6	1	0	2	1
t_7	2	1	0	1
t_8	2	0	2	0

R_2	A	C	F
t_1	0	0	0
t_2	0	1	1
t_3	0	2	1
t_4	1	1	1
t_5	1	2	2
t_6	2	0	2
t_7	2	2	1

R_3	B	C	E
t_1	0	0	0
t_2	0	1	0
t_3	0	1	1
t_4	0	2	1
t_5	1	0	2
t_6	1	1	2
t_7	1	2	2



Remove the block from tuples to check

A Few Notes

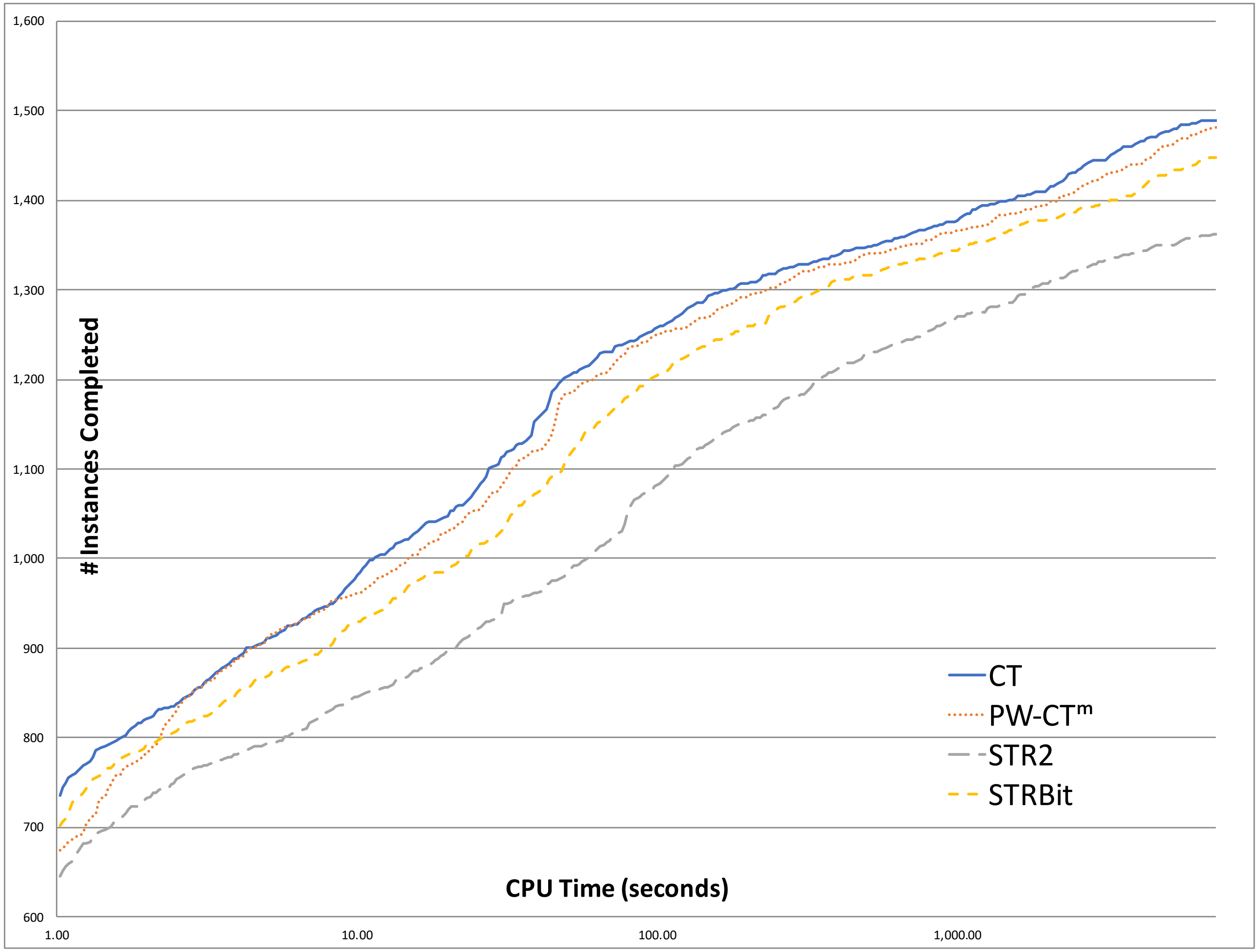
- At preprocessing
  - Iterate once over subscopes to partially enforce PWC
- At every step, including preprocessing, loop
  - Run CT first
  - Then only do one pass over PWC queue of relations
- Implementation requires additional data structures

… See paper for detailed explanations

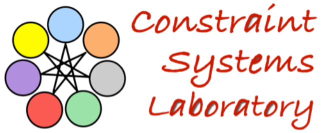
Experimental Setup

- Backtrack search, first solution, dom/ddeg, dom/wdeg
- GAC algorithms
  - CT, STR2, STRBit
- fPWC algorithms
  - eSTR2 and eSTR2^w (improved using minimal dual graph)
  - STRBit + PW-AC (improved using minimal dual graph, subscope)
  - PW-CT
- 2 hour timeout, 8 gigs per instance
- XCSP benchmark of CP Solver Competition

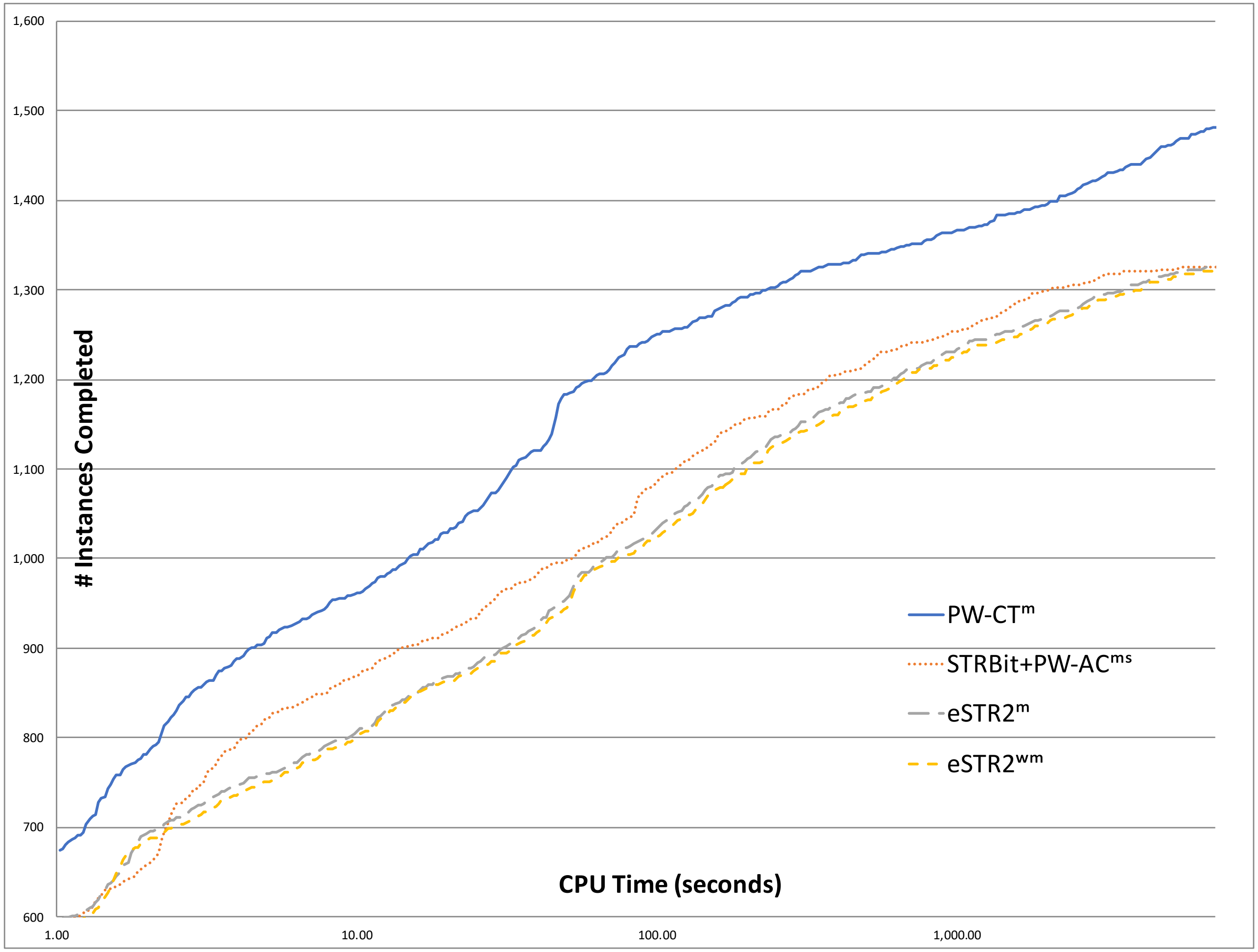
PW-CT vs GAC (dom/ddeg)



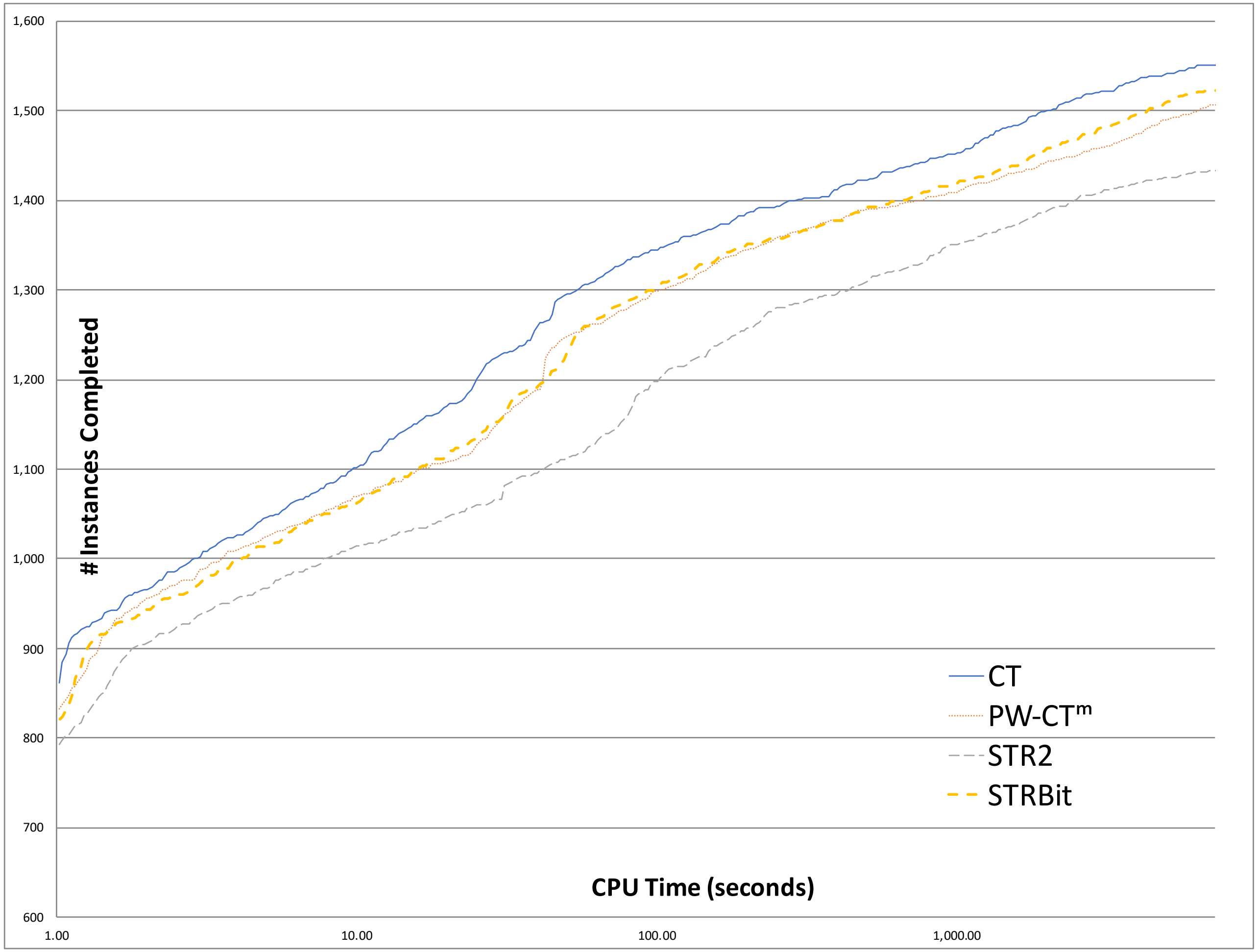
Constraint Systems Laboratory



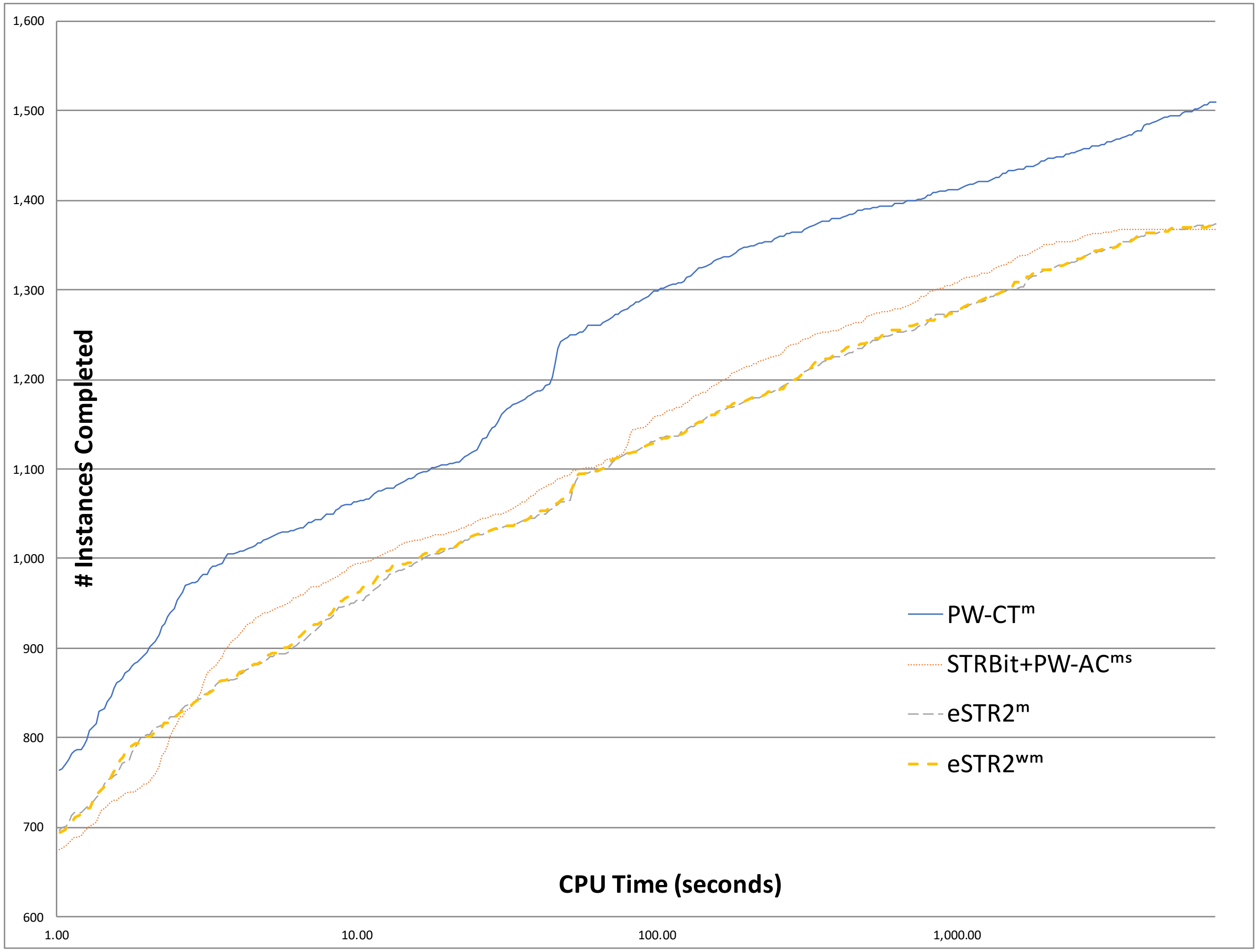
PW-CT vs PWC's (dom/ddeg)



PW-CT vs GAC (dom/wdeg)



PW-CT vs PWC's (dom/wdeg)



Detailed Results

Tested 2,210 non-binary instances

		dom/ddeg				dom/wdeg			
		#Cmpltd	ΣCPU (sec)	#MO	#NV	#Cmpltd	ΣCPU (sec)	#MO	#NV
71 Benchmarks tested with 2,210 total instances total									
GAC	CT	1,411	>449,421	65	2.90M	1,474	>338,246	65	1.65M
	STR2	1,284	>1,327,706	64	2.90M	1,355	>1,164,208	64	1.65M
	STRBit	1,370	>765,923	64	2.90M	1,445	>600,089	65	1.65M
fPWC	PW-CT	1,403	>579,112	65	0.99M	1,428	>715,885	65	0.82M
	PW-CT^m	1,403	>567,500	65	0.99M	1,431	>696,622	65	0.82M
	STRBit+PW-AC	1,213	>1,738,628	137	0.99M	1,263	>1,752,986	139	0.84M
	STRBit+PW-AC^{ms}	1,247	>1,472,804	113	0.99M	1,290	>1,527,538	113	0.84M
	eSTR2	1,231	>1,750,990	102	0.99M	1,282	>1,769,454	102	0.83M
	eSTR2^m	1,248	>1,588,847	99	0.99M	1,295	>1,627,281	99	0.83M
wPWC	eSTR2^w	1,227	>1,784,866	102	1.01M	1,280	>1,769,529	102	1.1M
	eSTR2^{wm}	1,243	>1,629,846	99	1.01M	1,294	>1,622,247	99	1.1M

Conclusions

- Identified four features that improve performance of fPWC algorithms
  1. Piecewise functionality
  2. Subscope-based reasoning
  3. Minimal dual graph
  4. Two conditions when GAC guarantees fPWC
- Integrated features in a single new algorithm: PW-CT
  - Outperforms other fPWC algorithms
  - Is competitive with modern GAC algorithms

Thank you for listening

Questions?