

Solving Difficult SAT Instances In The Presence of Symmetry

Fadi A. Aloul, Arathi Ramani

Igor L. Markov and Karem A. Sakallah
University of Michigan

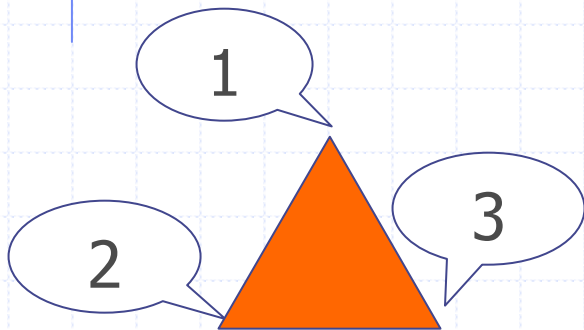
Highlights of Our Work

- ◆ No new SAT solvers are proposed
- ◆ We improve performance of existing complete SAT solvers *by preprocessing*
- ◆ Evaluate on carefully chosen SAT benchmarks
 - ignore easy benchmarks
 - only worry about benchmarks with symmetries (but the symmetries may not be given!)
 - show applicability to chip layout (200x speed-ups) and derive new hard SAT benchmarks
 - show asymptotic improvements

Outline

- ◆ Symmetries and permutations
 - Compact representations of symmetries
 - Computational group theory
- ◆ Symmetries of CNF instances
 - Detection via Graph Automorphism
 - Syntactic versus semantic symmetries
 - Using symmetries to speed up search
- ◆ Opportunistic symmetry detection
- ◆ Empirical results

Symmetries and Permutations



Symmetries of the triangle:

- $1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow 1$ (123)
- $1 \rightarrow 3, 3 \rightarrow 2, 2 \rightarrow 1$ (132)
- $1 \rightarrow 2, 2 \rightarrow 1, 3 \rightarrow 3$ (12)
- $1 \rightarrow 1, 2 \rightarrow 3, 3 \rightarrow 2$ (23)
- $1 \rightarrow 3, 3 \rightarrow 1, 2 \rightarrow 2$ (13)
- $1 \rightarrow 1, 2 \rightarrow 2, 3 \rightarrow 3$ "do nothing"

Cycles must be disjoint

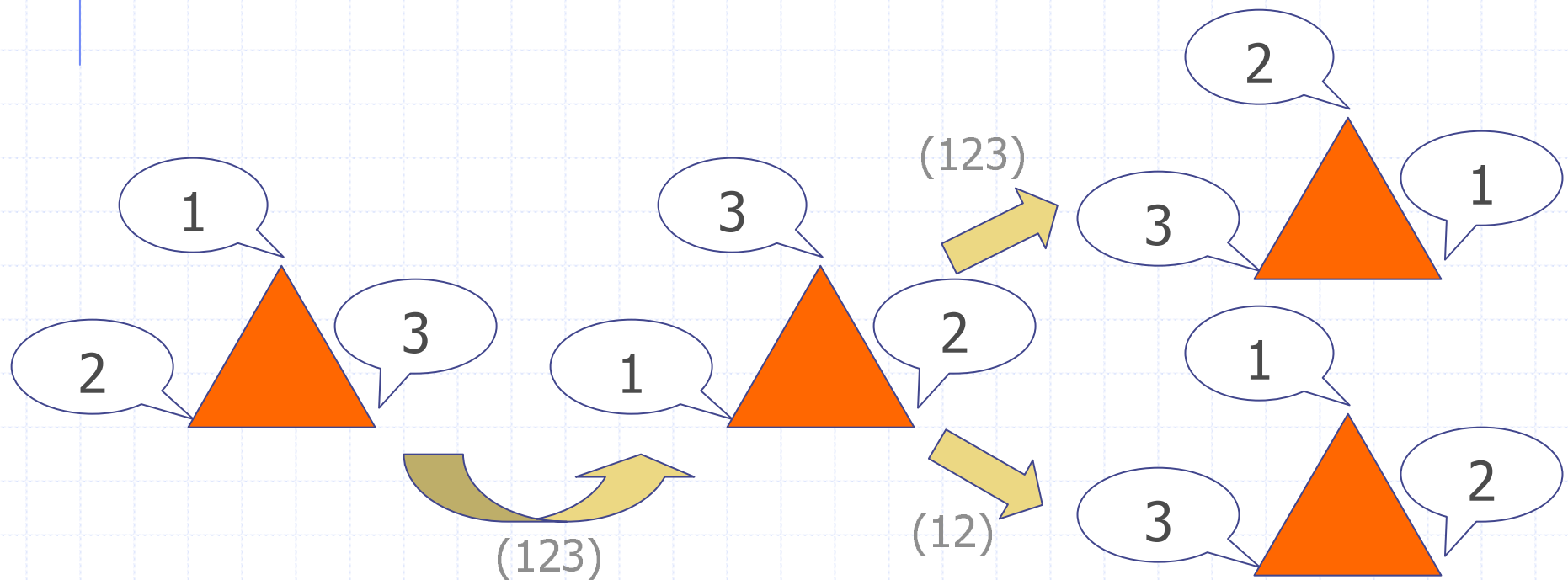
Symmetries and Permutations (2)

apply (123) and then again (123): get (132)

apply (123) and then (12) : get (23)

all non-trivial symmetries

are products of (123) and (12) - "generators"



Symmetries and Permutations (3)

- ◆ Idea: represent symmetries of an object by permutations that preserve the object
- ◆ Composition of symmetries is modeled by composition of permutations
 - Composition is associative
 - Every symmetry has an inverse
 - The do-nothing symmetry is the identity
- ◆ This enables applications of group theory

Compact Representations

- ◆ Represent the group of all symmetries
 - Do not list individual symmetries
 - List generating permutations (generators)
- ◆ Elementary group theory proves:
 - If redundant generators are avoided,
 - A group with N elements can be represented by at most $\log_2(N)$ generators
- ◆ Guaranteed exponential compression

Compact Representations (2)

- ◆ Sometimes can do better than $\log_2(N)$
- ◆ E.g., consider the group S_k of all $k!$ permutations of $1..k$
 - Can be generated by (12) and $(123..k)$
 - Or by $(12), (23), (34), \dots, (k-1 k)$
- ◆ To use this guaranteed compression, we need algorithms in terms of permutation generators

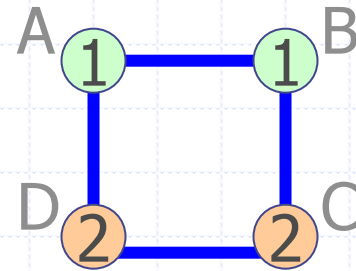
Computational Group Theory

- ◆ Algorithms for group manipulation in terms of generators are well known
 - Published by Sims, Knuth, Babai and others
 - Especially efficient for permutation groups
- ◆ High-quality implementations available
 - The GAP package – free, open-source (GAP="Groups, Algebra, Programming")
 - The MAGMA package – commercial

Finding Symmetries of Graphs

- ◆ A symmetry (automorphism) of a graph

- Permutation of vertices that maps edges to edges



- ◆ Additional constraints

- Vertex colors (labels): integers
- Every vertex must map into a vertex of same color

- ◆ Computational Graph Automorphism

- Given a graph
- Find generators of its group of symmetries
- $\text{GraphAuto} \in \text{NP}$, and is believed to $\notin \text{P}$ and $\notin \text{NPC}$
- Linear average-case runtime (but that's irrelevant!)
- Algorithms implemented in GAP(GRAPE(NAUTY))

Symmetries of CNF Formulae

- ◆ Permutations of variables that map clauses to clauses
 - E.g., $(a+b+c)(d+e+f)$ has one non-trivial symmetry $(ad)(be)(cf)$
 - Considering single swaps only is not enough
- ◆ Phase changes, e.g., $a \rightarrow a'$, and compositions with permutations
 - E.g., $(a+b+c)(d+e'+f')$ has one non-trivial symmetry $(ad)(be')(cf')$

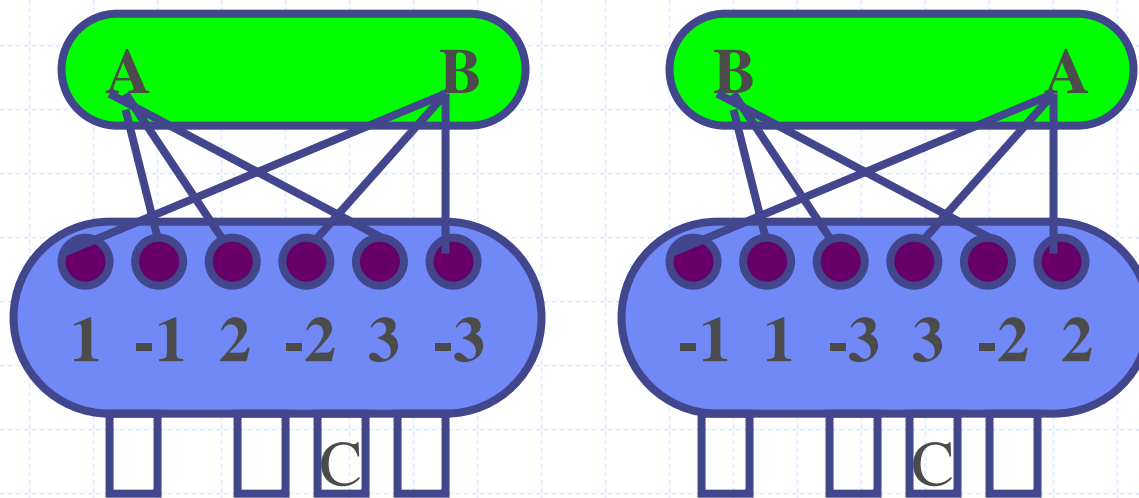
Reduction to Graph Automorphism

- ◆ CNF formula \rightarrow colored graph
 - Linear time and space
- ◆ Find graph's [colored] symmetries
 - Worst-case exponential time
- ◆ Interpret graph symmetries found as symmetries of the CNF formula
 - Permutational symmetries
 - Phase-shift symmetries

Reduction to Graph Automorphism

- ◆ Vertices of two colors: clauses and vars
 - One vertex per clause, two per variable
- ◆ Edges of three types: (i) incidence, (ii) consistency, and (iii) 2-literal clauses

Clauses: A (-1 + 2 + 3), B (1 + -2 + -3), C (-2 + 3)



1	2	3	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Syntactic and Semantic Symmetries

- ◆ CNF formula versus Boolean function
- ◆ Syntactic symmetries
 - symmetries of representation
- ◆ Semantic symmetries of the object
 - E.g., permutations and negations of variables that preserve the value of the function for all inputs
- ◆ Any syntactic symmetry is also semantic
 - but not vice versa, example: $(a)(a')(a+b)$

Speeding up SAT Search

- ◆ Search space may have symmetries
 - May have regions that map 1:1
 - This makes search redundant
- ◆ Ideas for speed-ups
 - Consider equivalence classes under symmetry
 - Pick a representative for each class
 - Search only one representative per class
- ◆ This restricted search is \Leftrightarrow to original

Symmetry-breaking Predicates

- ◆ To restrict search
 - Add clauses to the original CNF formula (“symmetry-breaking” clauses)
 - They will pick representatives of classes and restrict search
- ◆ Our main task is to find those clauses
 - Every permutation \rightarrow group of clauses (a “symmetry-breaking” predicate)
 - Use only generators of semantic symmetries

Construction of S.-b. Predicates

◆ Earlier work:

- By Crawford, Ginsberg, Roy and Luks (92,96)
- Not based on cycle notation for permutations

CGRL

◆ Our construction is more efficient

- Every cycle considered separately
- In practice almost all cycles are 2- or 3-cycles
 - ◆ Two types of 2-cycles: (aa') and (ab)
 - ◆ Symm.-breaking predicates: (a) and $(a'+b)$ resp.
- For multiple cycles
 - ◆ Procedure to chain symmetry-breaking predicates

Details: Individual Cycles

- ◆ Use an ordering of all variables (arbitrary)
 - To prevent transitivity violations: $(a+b')(b+c')(c+a)$
(the construction by CGRL uses an ordering as well)
- ◆ Symmetry-breaking predicate for cycle (ab):
 - $(a \Rightarrow b)$ aka $(a \leq b)$, if a precedes b in the ordering
 - Think of partial variable assignments to b and a
 - ◆ Must choose one from 01 and 10
- ◆ S.-b. predicate for cycle (abc) is $(a \leq b \leq c)$
 - For 3-var partial assignments, can cycle all 0s to front
- ◆ For longer cycles, still can improve upon CGRL
- ◆ Does ordering affect overall performance?

Details: Multiple Cycles(1)

◆ Solution space reduction

- By **2x** when (a) is added to break cycle (aa')
- Still by **2x** if permutation has cycles (aa') and (bb')
- By **4/3x** when (a'+b) is added to break cycle (ab)
- **What if** a permutation has cycles (ab) and (cd) ?
- By **2x** when (a≤b≤c) is added to break (abc)

◆ Suppose you have cycles (aa') and (uvt)

- Adding both predicates cuts solution space by **4x**

◆ Rule of thumb: after breaking a 2-cycle, symmetry-break the square of the permutation

Details: Multiple Cycles(2)

- ◆ Rule of thumb: after breaking a 3-cycle, symmetry-break the cube of the permutation
- ◆ What if we have both (xy) and (uv) ?
 - Squaring will kill the second cycle, so don't square!
 - Look at partial assignments for x,y : 00, 01, 10 and 11
 - For 10 or 01, $(x'+y)$ is all we can do
 - For 00 or 11, can add $(u'+v)$
 - Adding $(x \leq y)$ and $(x=y) \Rightarrow (u \leq v)$ cuts the solution space by **8/5x** (better than **4/3x**)
- ◆ For 3-cycles, add $(x=y=z) \Rightarrow (u \leq v \leq w)$ or the like
- ◆ For multiple cycles $((x=y=z) \& (a=b)) \Rightarrow (u \leq v)$, etc

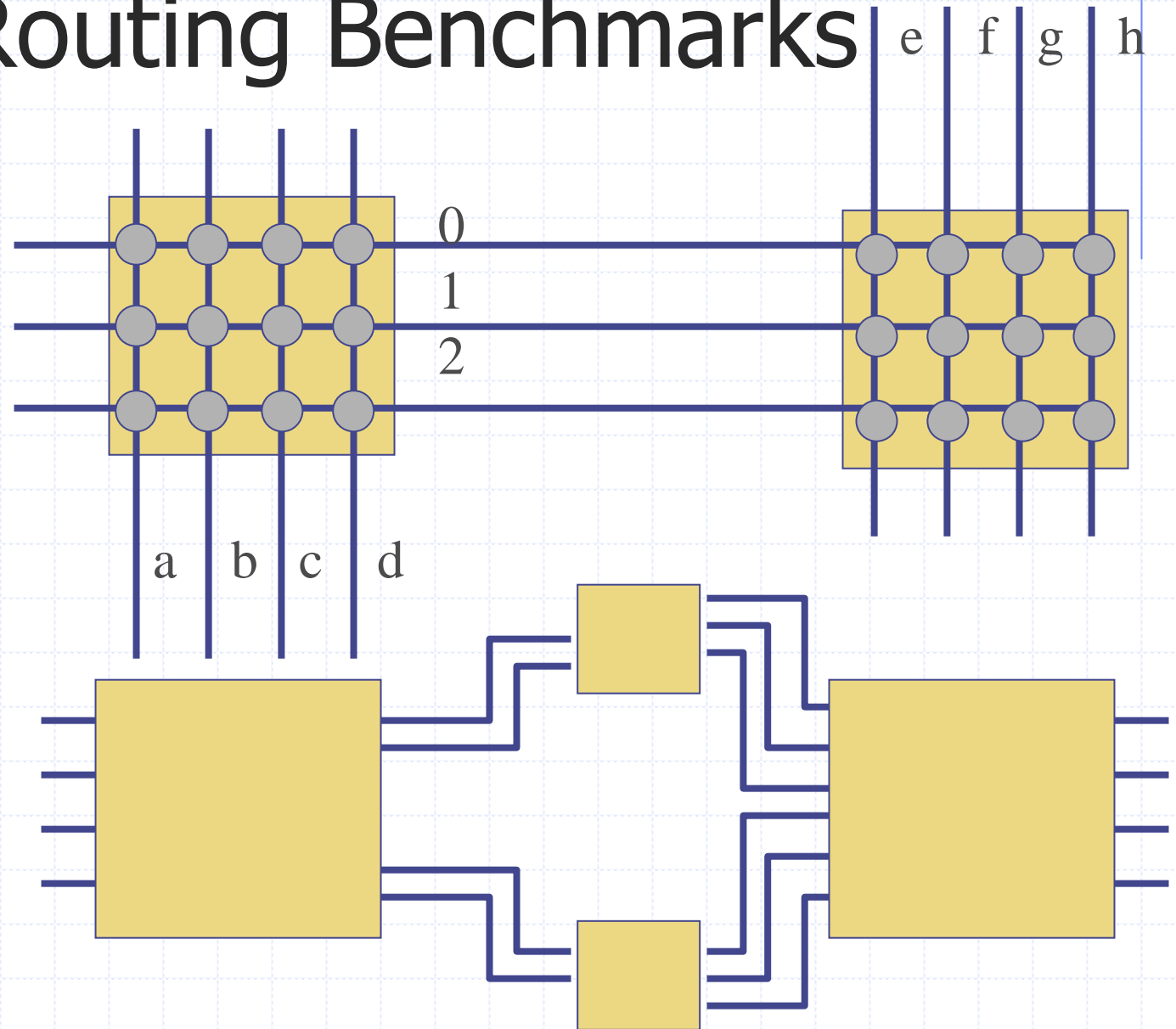
Discussion

- ◆ We detect syntactic symmetries only
 - If more semantic symmetries available, can use them in the same way
- ◆ Symmetry-detection can take long time
 - Sometimes longer than solving SAT
- ◆ In some cases the only symmetry is trivial
 - Symm. detection is often fast in these cases
- ◆ Symmetry-breaking using generators only is not exhaustive (remark by CGRL)
 - But makes symmetry-breaking practical (our result)
 - Pathological cases are uncommon:why?(future work)

Evaluation and Benchmarks

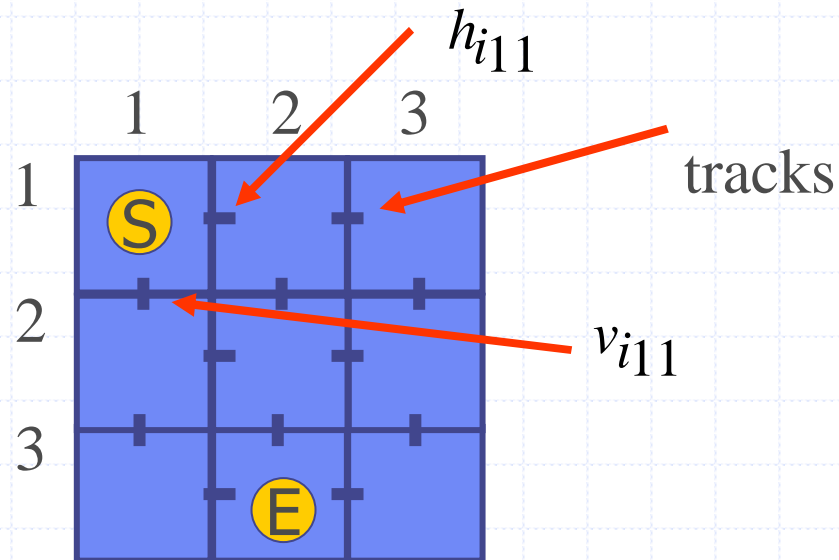
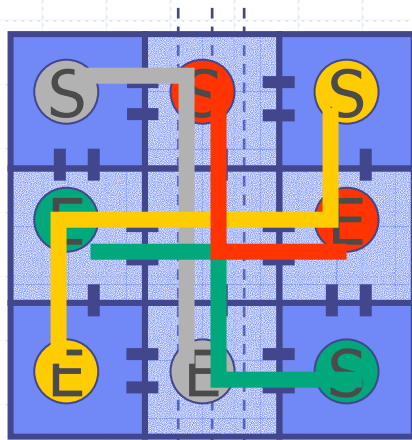
- ◆ Most of DIMACS benchmarks are easy for existing solvers
- ◆ We focus on difficult CNF instances
 - Pigeon-hole-n (PHP-n), Urquhart, etc.
- ◆ Observe that PHP-n can appear in apps
- ◆ EDA layout apps (routing) → symmetry
- ◆ We generate satisfiable and unsatisfiable CNF instances related to PHP-n

FPGA Routing Benchmarks



Global Routing Benchmarks

- ◆ Construct difficult grid-routing instances by “randomized flooding”
- ◆ Then convert to CNF



Empirical Results - Chaff

Instance	S/U	#V	#CL	Plain Chaff sec	Time- out %	Symmetries				Speedup		
						Finding sec	Number of	#generators cycles	Search Time	Total	Search only	
hole7	U	56	204	0.37	0%	0.1	2.03E+08	all	13	0.01	3.32	36.50
hole8	U	72	297	1.27	0%	0.07	1.46E+10	all	15	0.01	15.22	94.15
hole9	U	90	415	3.79	0%	0.1	1.32E+12	all	17	0.02	32.00	204.97
hole10	U	110	561	22.44	0%	0.15	1.45E+14	all	19	0.02	130.07	997.18
hole11	U	132	738	212.73	0%	0.13	1.91E+16	all	21	0.03	1329.54	7090.88
hole12	U	156	949	1000	100%	0.24	2.98E+18	all	23	0.04	3597.12	26315.79
Urq3_5	U	46	470	232.44	10%	0.48	5.37E+08	all	29	0.00	484.16	2.32E+06
Urq4_5	U	74	694	250.01	25%	1.35	8.80E+12	all	43	0.00	185.18	2.50E+06
Urq5_5	U	121	1210	1000	100%	13.15	4.72E+21	all	72	0.00	76.05	1.00E+07
Urq6_5	U	180	1756	1000	100%	62.93	6.49E+32	all	109	0.00	15.89	1.00E+07
Urq7_5	U	240	2194	1000	100%	176.62	1.12E+43	all	143	0.00	5.66	1.00E+07

Empirical Results - Chaff

Instance	S/U	#V	#CL	Plain Chaff sec	Time - out %	Symmetries				Speedup	
						Finding sec	Number of	#generators cycles	Search Time	Total	Search only
grout3.3-01	S	864	7592	19.01	0%	4.79	8.71E+09	10 26	0.67	3.48	28.37
grout3.3-03	S	960	9156	44.35	0%	8.94	6.97E+10	10 29	0.40	4.75	110.89
grout3.3-04	S	912	8356	19.36	0%	6.81	2.61E+10	10 27	0.36	2.70	53.79
grout3.3-08	S	912	8356	21.30	0%	7.14	3.48E+10	10 28	0.67	2.73	31.80
grout3.3-10	S	1056	10862	28.18	0%	10.65	3.48E+10	10 28	0.85	2.45	33.15
chnl10x11	U	220	1122	22.17	0%	0.45	4.20E+28	all 39	0.11	39.91	210.13
chnl10x12	U	240	1344	81.88	0%	0.61	6.04E+30	all 41	0.12	111.63	663.00
chnl10x15	U	300	2130	657.61	25%	1.28	4.50E+37	all 47	0.17	454.78	3961.49
chnl11x12	U	264	1476	207.37	0%	0.75	7.31E+32	all 43	0.15	231.31	1415.51
chnl11x13	U	286	1742	788.32	20%	1.08	1.24E+35	all 45	0.16	633.45	4792.24
chnl11x20	U	440	4220	1000	100%	4.4	1.89E+52	all 59	0.31	212.49	3267.97

Empirical Results - Chaff

Instance	S/U	#V	#CL	Plain	Time	Symmetries				Speedup		
				Chaff	out	Finding	Number	#generators	Search	Total	Search	
				sec	%	sec	of	cycles	Time		only	
fpga10_8	S	120	448	7.56	0%	0.63	6.00E+71	all	62	0.05	11.15	157.56
fpga10_9	S	135	549	3.80	0%	0.88	6.33E+77	all	68	0.03	4.16	113.39
fpga12_11	S	198	968	694.00	50%	3.76	7.18E+77	all	95	0.06	181.63	11377.05
fpga12_12	S	216	1128	80.20	0%	5.31	7.44E+77	all	104	0.13	14.74	616.92
fpga12_8	S	144	560	246.70	10%	1.23	8.41E+77	all	72	0.08	188.39	3103.14
fpga12_9	S	162	684	885.00	80%	1.7	2.25E+77	all	79	0.05	504.56	16388.89
fpga13_9	S	176	759	550.00	85%	2.57	2.56E+77	all	84	0.06	208.81	8593.75
fpga13_10	S	195	905	1000	100%	4.04	5.76E+77	all	93	0.08	242.60	12195.12
fpga13_12	S	234	1242	1000	100%	6.9	8.85E+77	all	110	0.08	143.23	12195.12

Empirical Results - Chaff

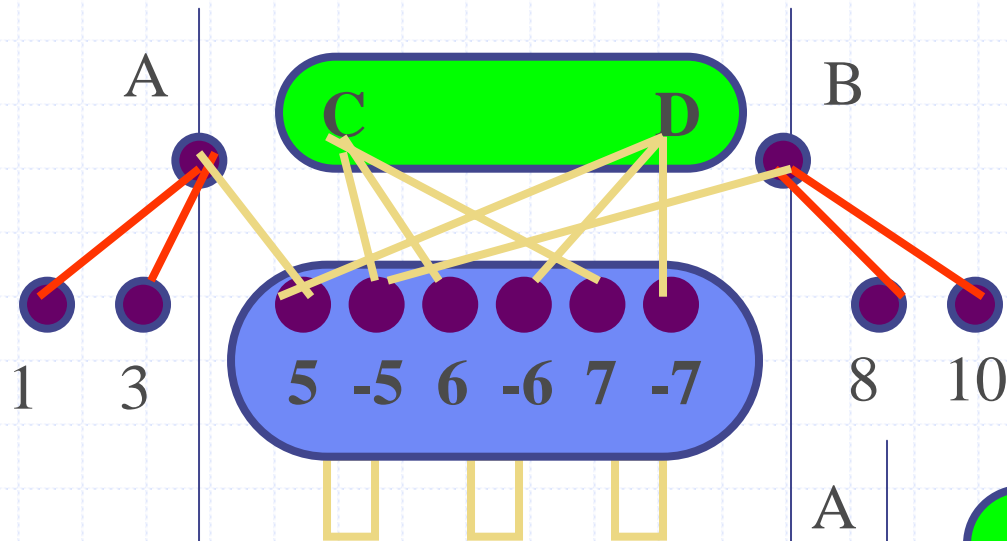
Instance	S/U	#V	#CL	Plain Chaff sec	Time- out %	Symmetries				Speedup		
						Finding sec	Number of	#generators cycle	Search Time	Tot	Search only	
2dlx_ca_mc	U	3250	24640	6.54	0%	38.36	9.36E+77	10	66	6.30	0.15	1.04
2pipe	U	892	6695	2.08	0%	10.74	2.26E+45	10	38	1.56	0.17	1.33
2pipe_1_ooo	U	834	7026	2.55	0%	9.37	8.00E+00	10	3	1.80	0.23	1.41
2pipe_2_ooo	U	925	8213	3.43	0%	11.14	3.20E+01	10	5	2.82	0.25	1.22
3pipe	U	2468	27533	36.44	0%	463.57	7.29E+77	10	85	19.65	0.08	1.85
2dlx_ca_mc	U	3250	24640	6.54	0%	3.17	2.34E+77	10	64	5.42	0.76	1.21
2pipe	U	892	6695	2.08	0%	10.47	2.26E+45	10	38	1.30	0.18	1.60
2pipe_1_ooo	U	834	7026	2.55	0%	9.02	8.00E+00	10	3	1.80	0.24	1.41
2pipe_2_ooo	U	925	8213	3.43	0%	11.09	3.20E+01	10	5	2.80	0.25	1.23
3pipe	U	2468	27533	36.44	0%	3.63	1.42E+77	10	78	36.20	0.91	1.01
4pipe	U	5237	80213	337.61	0%	9.32	1.03E+78	10	142	334.00	0.98	1.01
5pipe	U	9471	195452	325.92	0%	29.42	3.64E+78	10	227	290.50	1.02	1.12

Instance	S/U	#V	#CL	Plain Chaff sec	Time- out %	Symmetries					Speedup	
						Finding sec	Number of	#generators cycles	Search Time	Total	Search only	
grout3.3-01	S	864	7592	19.01	0%	4.79	8.71E+09	10	26	0.67	3.48	28.37
grout3.3-03	S	960	9156	44.35	0%	8.94	6.97E+10	10	29	0.40	4.75	110.89
grout3.3-04	S	912	8356	19.36	0%	6.81	2.61E+10	10	27	0.36	2.70	53.79
grout3.3-08	S	912	8356	21.30	0%	7.14	3.48E+10	10	28	0.67	2.73	31.80
grout3.3-10	S	1056	10862	28.18	0%	10.65	3.48E+10	10	28	0.85	2.45	33.15
chnl10x11	U	220	1122	22.17	0%	0.45	4.20E+28	all	39	0.11	39.91	210.13
chnl10x12	U	240	1344	81.88	0%	0.61	6.04E+30	all	41	0.12	111.63	663.00
chnl10x15	U	300	2130	657.61	25%	1.28	4.50E+37	all	47	0.17	454.78	3961.49
chnl11x12	U	264	1476	207.37	0%	0.75	7.31E+32	all	43	0.15	231.31	1415.51
chnl11x13	U	286	1742	788.32	20%	1.08	1.24E+35	all	45	0.16	633.45	4792.24
chnl11x20	U	440	4220	1000	100%	4.4	1.89E+52	all	59	0.31	212.49	3267.97
fpga10_8	S	120	448	7.56	0%	0.63	6.00E+71	all	62	0.05	11.15	157.56
fpga10_9	S	135	549	3.80	0%	0.88	6.33E+77	all	68	0.03	4.16	113.39
fpga12_11	S	198	968	694.00	50%	3.76	7.18E+77	all	95	0.06	181.63	11377.05
fpga12_12	S	216	1128	80.20	0%	5.31	7.44E+77	all	104	0.13	14.74	616.92
fpga12_8	S	144	560	246.70	10%	1.23	8.41E+77	all	72	0.08	188.39	3103.14
fpga12_9	S	162	684	885.00	80%	1.7	2.25E+77	all	79	0.05	504.56	16388.89
fpga13_9	S	176	759	550.00	85%	2.57	2.56E+77	all	84	0.06	208.81	8593.75
fpga13_10	S	195	905	1000	100%	4.04	5.76E+77	all	93	0.08	242.60	12195.12
fpga13_12	S	234	1242	1000	100%	6.9	8.85E+77	all	110	0.08	143.23	12195.12
2dlx_ca_mc	U	3250	24640	6.54	0%	38.36	9.36E+77	10	66	6.30	0.15	1.04
2dlx_ca_mc	U	3250	24640	6.54	0%	3.17	2.34E+77	10	64	5.42	0.76	1.21
2pipe	U	892	6695	2.08	0%	10.47	2.26E+45	10	38	1.30	0.18	1.60
2pipe_1_ooo	U	834	7026	2.55	0%	9.02	8.00E+00	10	3	1.80	0.24	1.41
2pipe_2_ooo	U	925	8213	3.43	0%	11.09	3.20E+01	10	5	2.80	0.25	1.23
3pipe	U	2468	27533	36.44	0%	3.63	1.42E+77	10	78	36.20	0.91	1.01
4pipe	U	5237	80213	337.61	0%	9.32	1.03E+78	10	142	334.00	0.98	1.01
5pipe	U	9471	195452	325.92	0%	29.42	3.64E+78	10	227	290.50	1.02	1.12

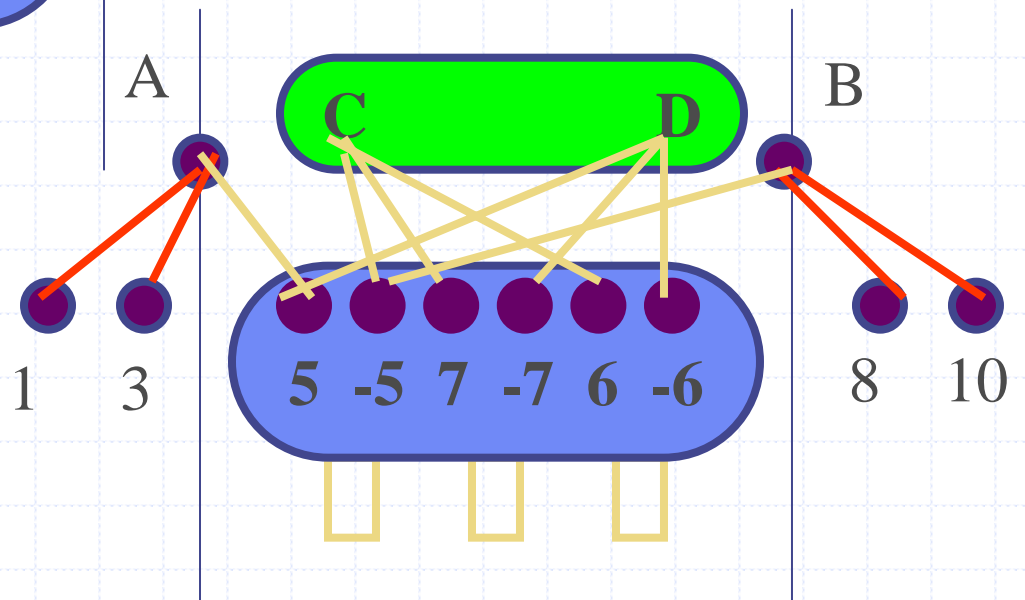
Domain-specific Symmetry-Breaking Predicates

- ◆ We looked at symmetry generators for global routing benchmarks
- ◆ Those symmetries were permutations of routing tracks
- ◆ Symmetry-breaking clauses can be added when converting to CNF
 - Serious speed-up for Chaff in all cases
- ◆ No symmetries left after that

Fast Symmetry Detection



$A (1 + 3 + 5)$
 $B (-5 + 8 + 10)$
 $C (-5 + 6 + 7)$
 $D (5 + -6 + -7)$



Conclusions

- ◆ Pre-processing speeds up SAT solvers on difficult instances with symmetries
 - Strong empirical results on new and old BMs
- ◆ Improved constructions
 - Reduction to graph automorphism
 - Symmetry-breaking predicates
 - ◆ Cycle-based construction
 - ◆ Using generators only
- ◆ Many important questions not answered
- ◆ Significant on-going work

TRANKU