Bipartite Perfect Matching Benchmarks

Cayden R. Codel, Joseph E. Reeves,
Marijn J. H. Heule, and Randal E. Bryant

Carnegie Mellon University

Introduction

The pigeonhole and mutilated chessboard problems are challenging benchmarks for most SAT solvers

Some solvers employ special techniques that efficiently solve the canonical versions of these two problems

We extend the problems with randomized constructions and various encodings to evaluate specific solvers and encourage robust implementations

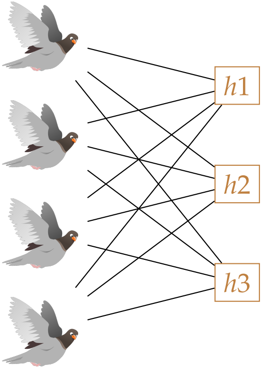
We also explore the impact of symmetry-breaking within this problem space

Bipartite Problems and Encodings

Random Bipartite Graphs

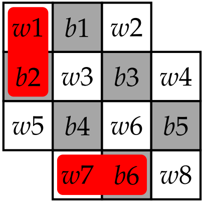
Symmetry-breaking

Pigeonhole Problem (PHP)

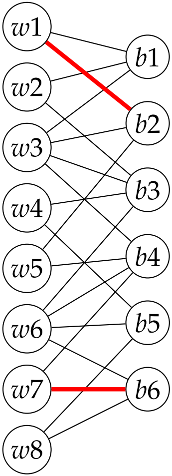


- Place $n + 1$ pigeons into n holes
- Fully connected $K_{n,n+1}$
- Resolution proofs exponential

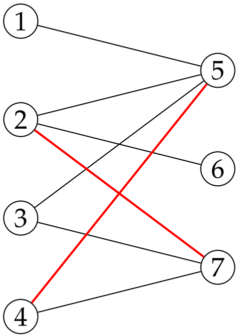
Mutilated Chessboard Problem (MChess)



- Tile an $n \times n$ board missing corners
- Partition black and white squares
- Dominoes are edges
- Resolution proofs exponential



Random Bipartite Graphs

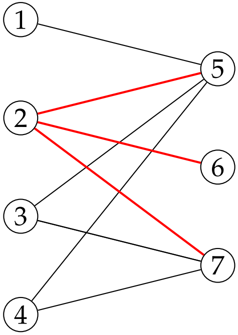


- Start with random spanning tree (**black**) on $n \times m$ partitions
- Add edges (red) to desired

  $$density = \frac{\#edges}{n \times m}$$

- Cardinality $(n - m)$ set to 1 for experiments

Encoding as a CNF



ALO (n_2) $e_{2,5} \vee e_{2,6} \vee e_{2,7}$

- Satisfying assignment is edges in perfect matching
- At Least One (ALO)
- At Most One (AMO) - Pairwise, Sinz, Linear
- **Sparse**, ALO larger partition, AMO smaller partition (PHP)
- **Full**, ALO, AMO both partitions (redundant clauses)

Solvers

Kissat

- State-of-the-art CDCL solver
- Not especially tuned for these problem instances

Lingeling

- CDCL solver with focus on pre-processing
- Built-in cardinality resolution
- Similar tools found in SAT4J

SaDiCaL

- Satisfaction-driven clause learner
- Learns PR clauses based on "positive reducts"
- Hand-crafted PHP and MChess proofs

PGBDD

- Binary Decision Diagram (BDD)-based solver generating extended resolution proofs
- Hand-crafted schedules for PHP and MChess
- *Bucket elimination* for automatic solving

PGBDD-Sched

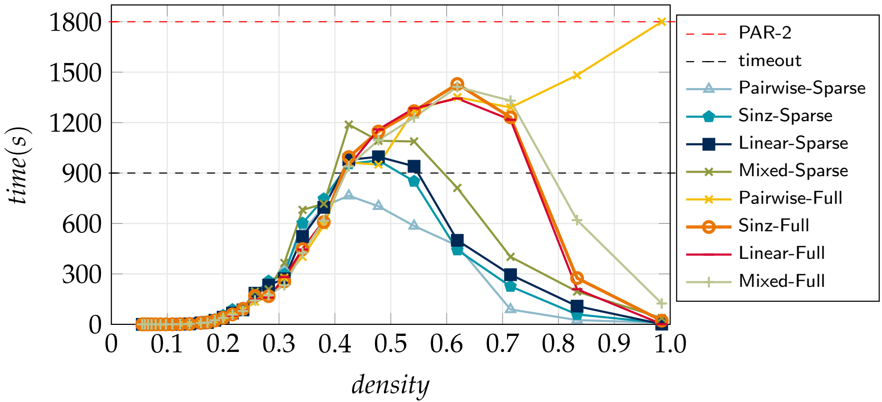
- Extends PGBDD with automation
- Generates variable and bucket orderings (schedules)
- Specific to grid structure of the Sinz encoding

Bipartite Problems and Encodings

Random Bipartite Graphs

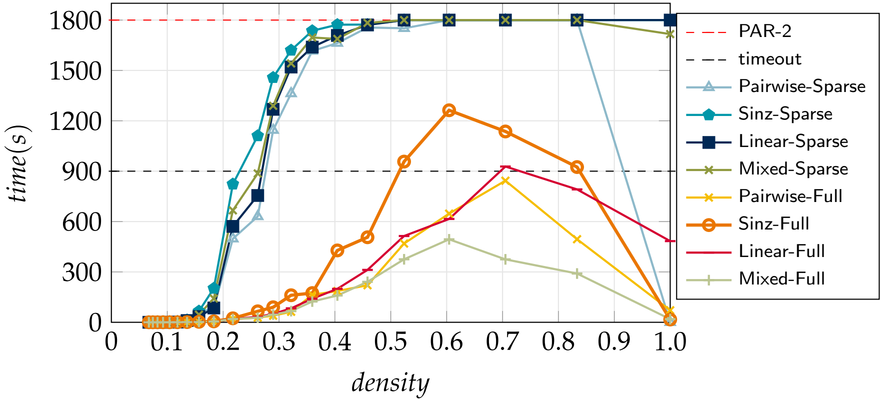
Symmetry-breaking

KISSAT on Random Bipartite Graphs with 130 Edges



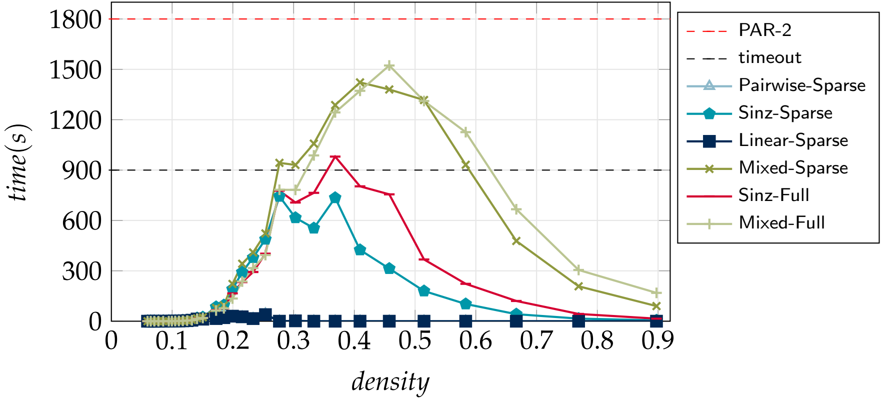
- 900 second timeout, 1800 second PAR-2
- Averaged over 60 seeds
- Sparse and Full encodings grouped together
- Mixed generally worse
- Pairwise-Full problem at higher density

SaDiCaL on Random Bipartite Graphs with 110 Edges



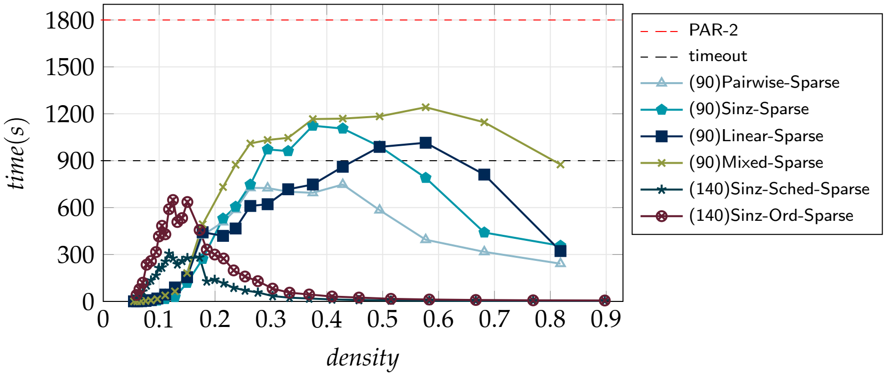
- Sparse encodings do terrible
- Mixed generally better for Full encodings
- Pairwise-Sparse with $density = 1$ is PHP

LINGELING on Random Bipartite Graphs with 140 Edges



- Absent experiments similar to Linear-Sparse
- Mixed and Sinz not detected in pre-processing
- AMO encodings grouped together, not Sparse and Full like KISSAT, i.e., resistant to redundant clauses

PGBDD on Random Bipartite Graphs with 90/140 Edges



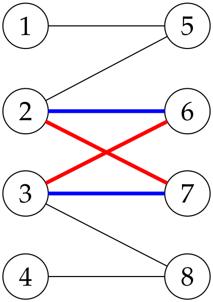
- Difference in edge count shows PGBDD general weakness
- A little information (variable or schedule ordering) helps a lot
- Best solver performance on mid range densities at 140 edges

Bipartite Problems and Encodings

Random Bipartite Graphs

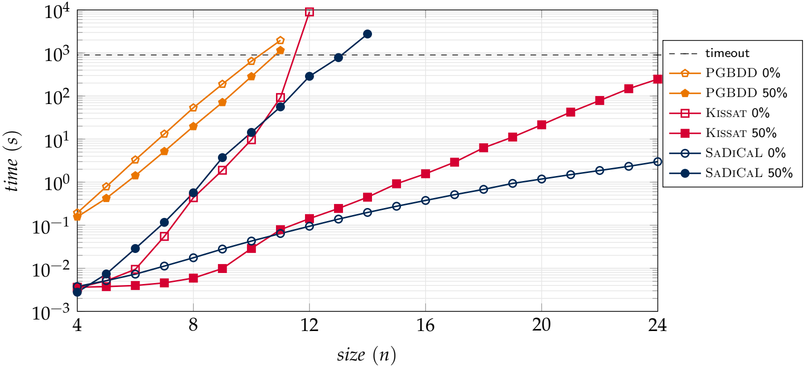
Symmetry-breaking

Symmetry Breaking Clauses in Bipartite Graphs



$\bar{e}_{2,7} \vee \bar{e}_{3,6}$ disallows the red matching in place of the blue matching

Solver Performance on Normal and Symmetry-broken PHP



- Symmetry-breaking clauses help KISSAT but hurt SADICAL
- Does little for brute-force approach of PGBDD

Conclusion and Future Work

Structured benchmark generators can be useful in evaluating and improving special purpose solvers

Future Work:

- Implement harder benchmarks for improving general solver performance
- Evaluate different types of symmetry-breaking clauses and their relation to PR clauses used in SaDiCaL
- Extend PGBDD-Sched to other problem domains that contain some underlying graph structure

AMO Encodings

Pairwise

AMO$(x_1, ..., x_n)$ is encoded as the conjunction of $(\overline{x}_i \vee \overline{x}_j)$ with $1 \leq i < j \leq n$

Sinz - introduce signal variables that propagate the AMO condition

$\overline{x}_i \vee s_i$ for $1 \leq i \leq n$ $\quad$ $\overline{s}_i \vee s_{i+1}$, $\overline{s}_i \vee \overline{x}_{i+1}$ for $1 \leq i < n$

Linear - introduce variables to split up Pairwise encoding when $n > 4$

Pairwise$(x_1, x_2, x_3, y) \wedge$ AMO$(\overline{y}, x_4, .., x_n)$