On Symbolic Approaches for Computing the Matrix Permanent

Supratik Chakraborty, **Aditya A. Shrotri**,
Moshe Y. Vardi

CP 2019

U. Conn., Stamford

Constrained Counting

"How many solutions does a set of constraints have?"

Constraint Types:

- All-Different, Exact-One, Cardinality, Parity …

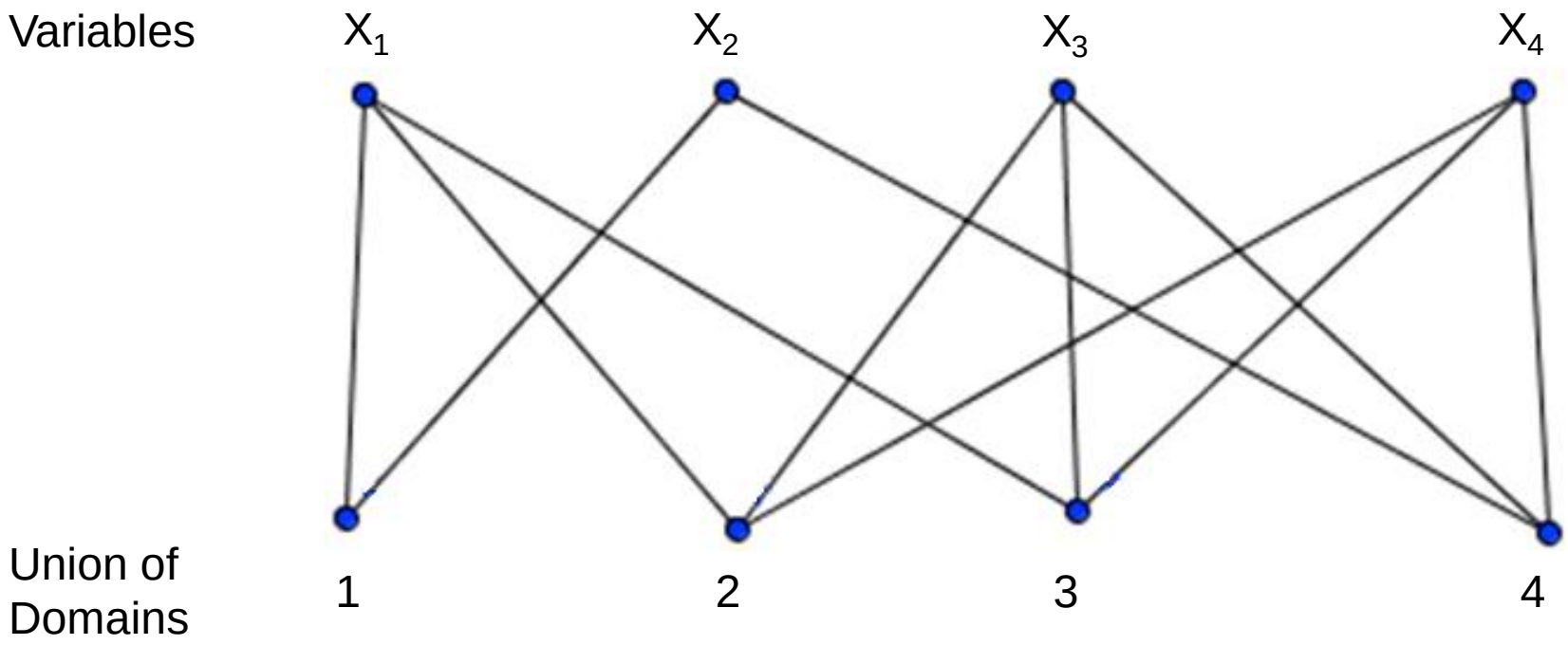
Applications of Counting:

- Counting-Based Branching Heuristics, Bayesian Inference, Verification, Computing Partition Functions, …

Counting-Based Branching Heuristics

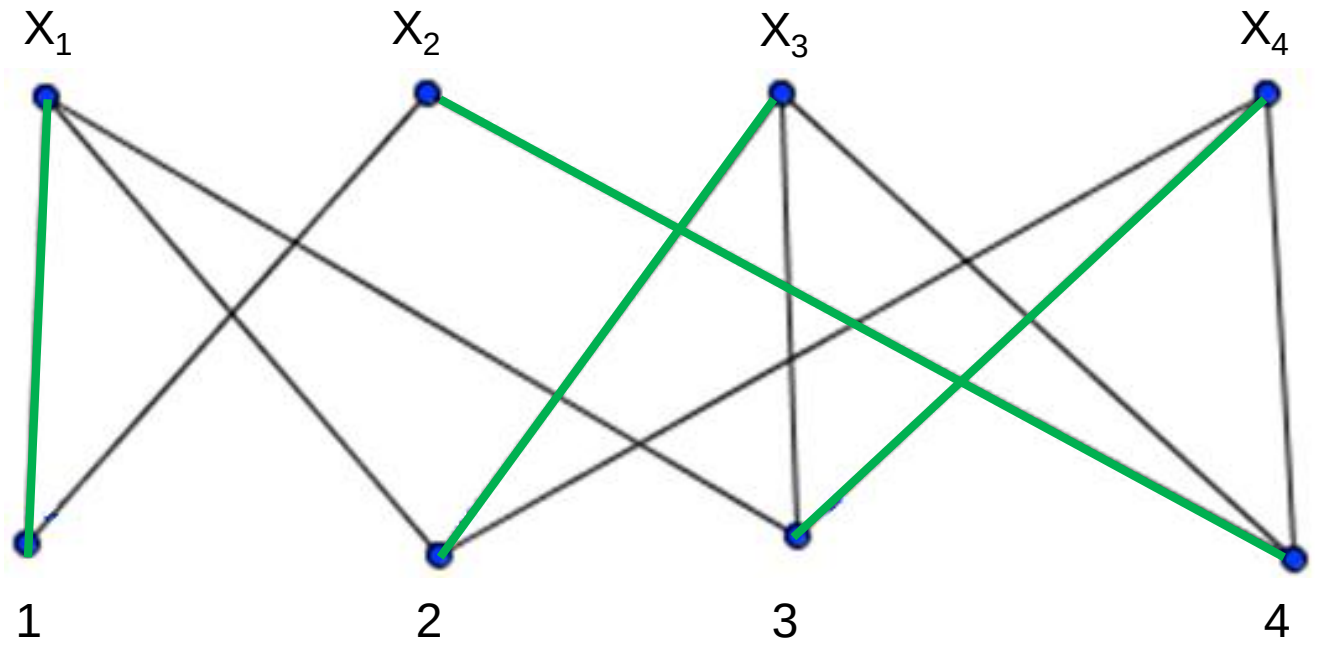
- Context: Solving CSPs
- Problem: Picking a variable and value to branch on
- Counting-Based Approach [Pesant,'05][Zanarini & Pesant, '09][Quimper et al., '12]
  - Focus search on constraint with least solutions (Fail-First)
    - Ex: minSC
  - Pick variable and value which preserves most solutions
    - Ex: maxSD

All-Different Constraint



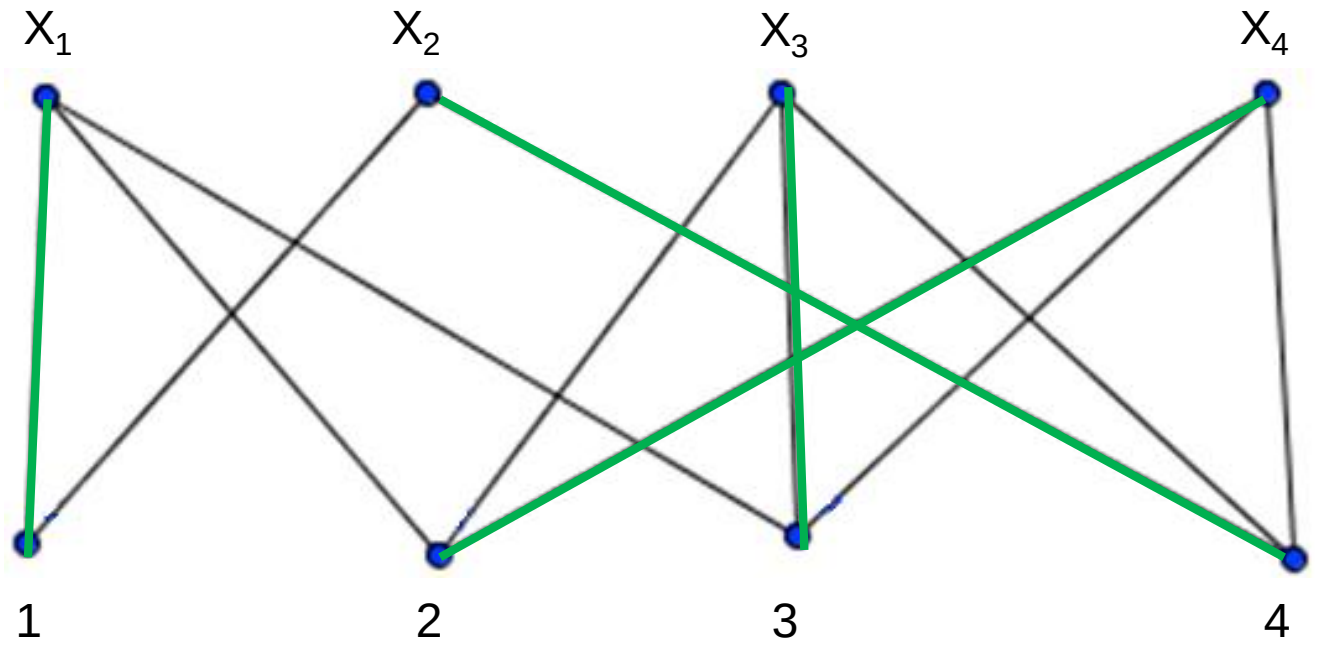
Task: Assign unique value to each variable

All-Different Constraint



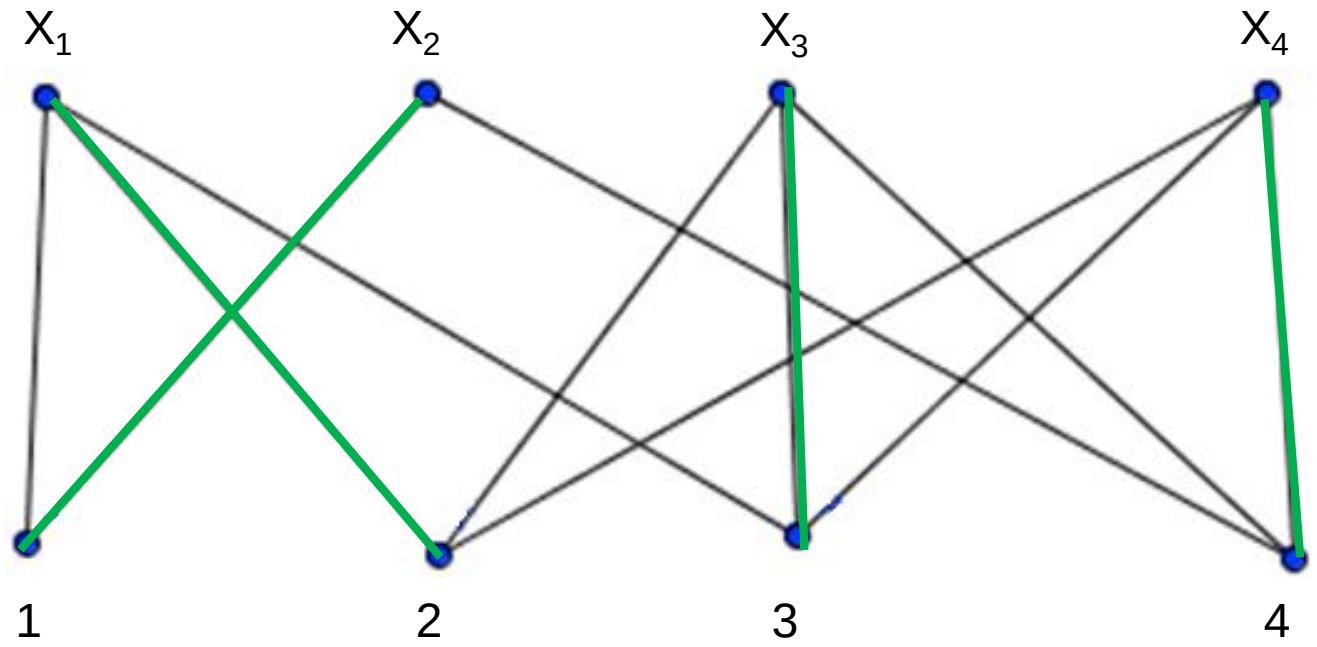
Solution: Bipartite Perfect Matchings!

All-Different Constraint



#Solutions = #Bipartite Perfect Matchings!

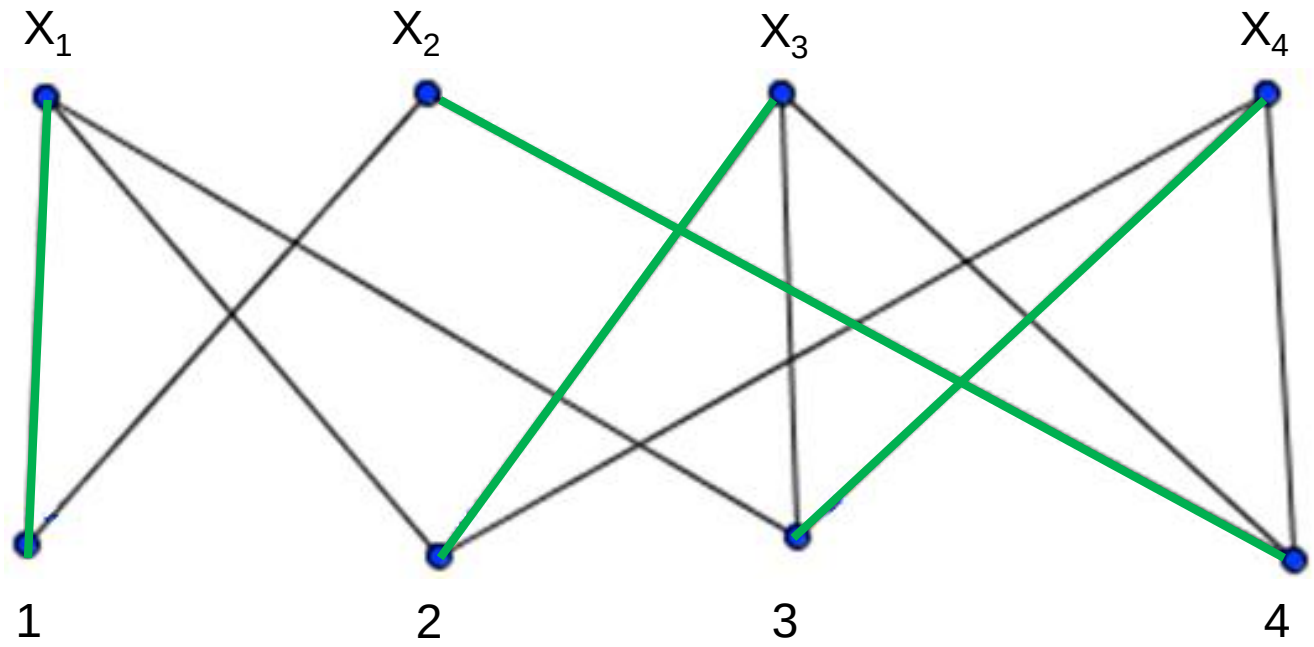
All-Different Constraint



#Solutions = #Bipartite Perfect Matchings!

Perfect Matchings and Matrix Permutations

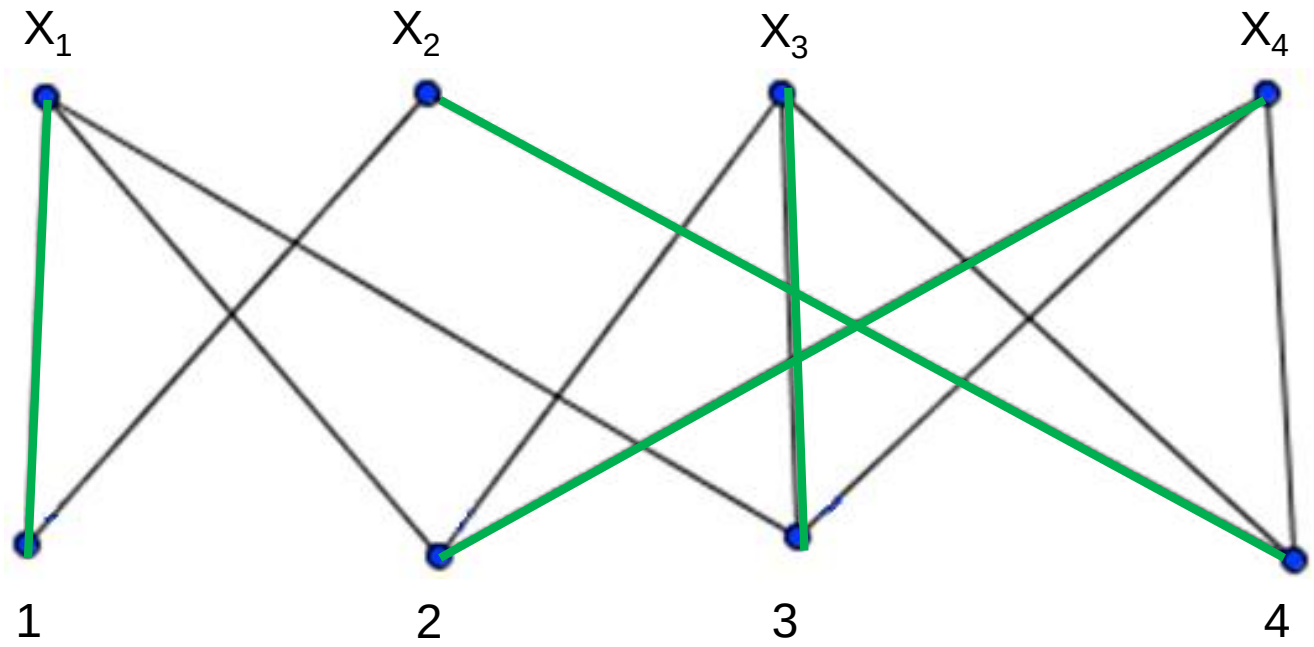
	1	2	3	4
X_1	1	1	1	0
X_2	1	0	0	1
X_3	0	1	1	1
X_4	0	1	1	1



#Solutions = # 1-permutations

Perfect Matchings and Matrix Permutations

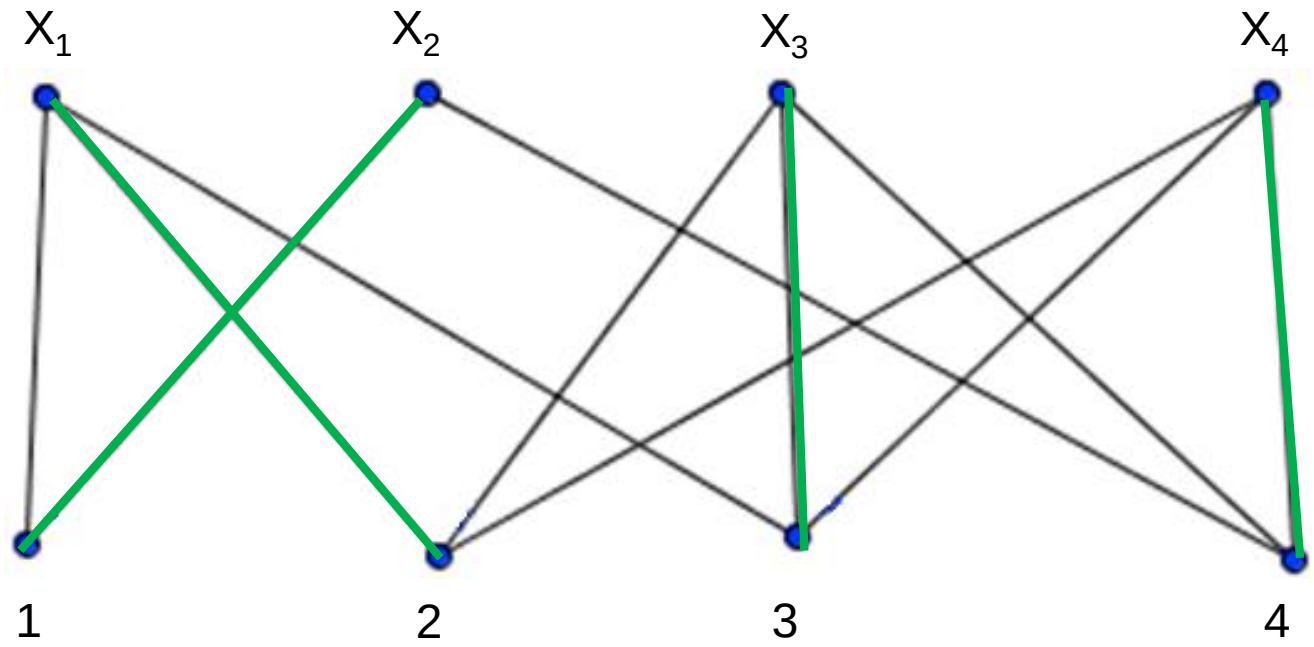
	1	2	3	4
X_1	1	1	1	0
X_2	1	0	0	1
X_3	0	1	1	1
X_4	0	1	1	1



#Solutions = # 1-permutations

Perfect Matchings and Matrix Permutations

	1	2	3	4
X_1	1	1	1	0
X_2	1	0	0	1
X_3	0	1	1	1
X_4	0	1	1	1



#Solutions = # 1-permutations

Permanent of a Matrix

- Given: $n \times n$ matrix $\boldsymbol{A} = (a_{i,j})$

$$perm(\boldsymbol{A}) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma_i}$$

- Ex:

$$\boldsymbol{A} = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

$$perm(\boldsymbol{A}) = aei + afh + bdi + bfg + cdh + ceg$$

Determinant of a Matrix

- Given: $n \times n$ matrix $\boldsymbol{A} = (a_{i,j})$

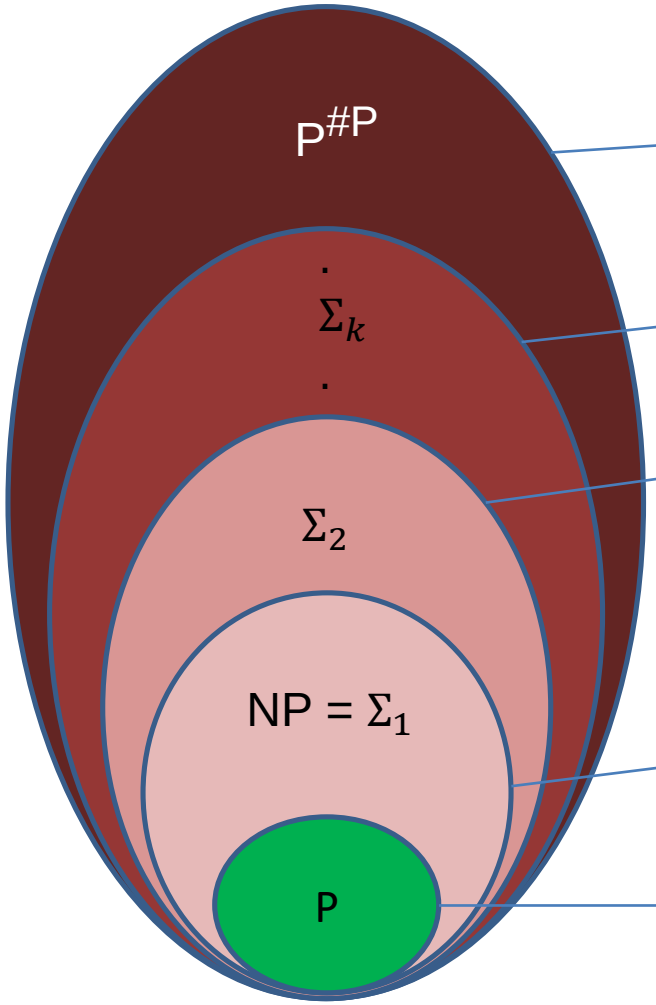
$$det(\boldsymbol{A}) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i}$$

- Ex:

$$\boldsymbol{A} = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

$$det(\boldsymbol{A}) = aei - afh - bdi + bfg + cdh - ceg$$

Hardness of Computing the Permanent



Permanent

$(\exists\forall)_k$ SAT, $(\text{Min-Max})_K$ CLIQUE…

∃∀SAT, MinMax CLIQUE, 3-Coloring Extension …

SAT, Max CLIQUE, 3-Colorability, Max Cut …

Determinant, Shortest Path, Min Cut, …

Prior Work

- Theory
  - Naïve Algorithm: $O(n!)$
  - Ryser's Formula: $\Theta(n \cdot 2^n)$ [Ryser, '63][Nijenhuis & Wilf '78]
  - Faster algorithms for special subclasses:
    - Sparse matrices [Izumi and Wadayama, '12][Cygan and Pilipczuk '15][Servedio & Wan,'05][Bjorklund and Williams,'19]
    - Bounded treewidth [Courcelle et al.,'00]
- Practice
  - Permanent of adjacency matrix of Fullerenes [Liang et al., '04,'06,'13,'15]
  - Benchmarking Classical Computing [Bjorklund et al., '18]

Contributions

- **RysersADD**: First algorithm to go beyond sparsity
  - Uses a symbolic approach
    - Algebraic Decision Diagrams [Bahar et al.,'97] + Early Abstraction [Dudek et al.,'18]
  - Works well on general class of matrices with "similar" rows
    - Includes both sparse and dense matrices as special cases

- Compelling empirical evidence of scalability
  - Dense Matrices: Vastly outperforms existing tools
  - Sparse matrices: Comparable to dedicated tools

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

- Select a subset S of columns

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

- Select a subset *S* of columns
- For each row, compute sum of entries in each of those columns

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

- Select a subset *S* of columns
- For each row, compute sum of entries in each of those columns
- Take product of the row-sums

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

- Select a subset *S* of columns
- For each row, compute sum of entries in each of those columns
- Take product of the row-sums
- Add / Subtract from total depending on size of *S*

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

$$perm(A) = (-1)^3 \cdot (\ldots)$$

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

$$perm(A) = (-1)^3 \cdot (-adg + \ \ldots)$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

$S = \{1\}$

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

$S = \{2\}$

$$perm(A) = (-1)^3 \cdot (-adg - beh + \cdots)$$

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

$S = \{3\}$

$$perm(A) = (-1)^3 \cdot (-adg - beh - cfi + \cdots)$$

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

$S = \{1,2\}$

$$perm(A) = (-1)^3 \cdot (-adg - beh - cfi + (a+b) \cdot (d+e) \cdot (g+h) + \ldots)$$

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

$S = \{1,3\}$

$$perm(A) = (-1)^3 \cdot (-adg - beh - cfi + (a+b) \cdot (d+e) \cdot (g+h) + (a+c) \cdot (d+f) \cdot (g+i) \ldots)$$

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

$S = \{2,3\}$

$$perm(A) = (-1)^3 \cdot (-adg - beh - cfi + (a+b) \cdot (d+e) \cdot (g+h) + (a+c) \cdot (d+f) \cdot (g+i) + (b+c) \cdot (e+f) \cdot (h+i) \ldots)$$

Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}$$

$$\begin{aligned} & perm(A) \\ & = (-1)^3 \cdot (-adg - beh - cfi + (a+b) \cdot (d+e) \\ & \cdot (g+h) + (a+c) \cdot (d+f) \cdot (g+i) + (b+c) \\ & \cdot (e+f) \cdot (h+i) - (a+b+c) \cdot (d+e+f) \\ & \cdot (g+h+i)) \end{aligned}$$

A Dynamic Programming Approach

- Memoize intermediate row-sums
  - Ex: Row-sums for S={1,2} can be reused when S={1,2,3}
  - Need to store $2^{|S|}$ row-sums in worst case

A Dynamic Programming Approach

- Memoize intermediate row-sums
  - Ex: Row-sums for S={1,2} can be reused when S={1,2,3}
  - Need to store $2^{|S|}$ row-sums in worst case

- Can we do better for special matrix classes?

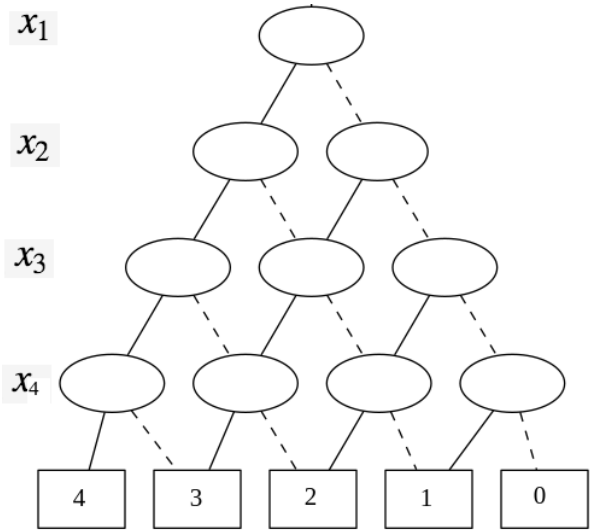
A Dynamic Programming Approach

- Memoize intermediate row-sums
  - Ex: Row-sums for S={1,2} can be reused when S={1,2,3}
  - Need to store $2^{|S|}$ row-sums in worst case

- Can we do better for special matrix classes?
  - Ideal Case: Identical Rows
    - Row-sums are identical for all S
  - In practice: Similar rows offer good compression

Algebraic Decision Diagrams

- Data Structures for compact representation of Real-valued Boolean functions $f: \{0,1\}^n \rightarrow R$
  - DAGs with fixed variable order and node-sharing
  - Operations: Sum, Product, Additive Quantification (∃), ITE
- Ex: $f = x_1 + x_2 + x_3 + x_4$



Representing Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

- Boolean variable x_j for each column j

Representing Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

- Boolean variable x_j for each column j
- $f_{RS}^{j} = x_{j_1} + x_{j_2} + \cdots + x_{j_k}$

Representing Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

- Boolean variable x_j for each column j
- $f_{RS}^{j} = x_{j_1} + x_{j_2} + \cdots + x_{j_k}$
- $f_{RSP} = \prod_{i=1}^{n} f_{RS}^{j}$

Representing Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

- Boolean variable x_j for each column j
- $f_{RS}^{j} = x_{j_1} + x_{j_2} + \cdots + x_{j_k}$
- $f_{RSP} = \prod_{i=1}^{n} f_{RS}^{j}$
- $f_{Parity} = (-1)^{|S|}$

Representing Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

- Boolean variable x_j for each column j
- $f_{RS}^{j} = x_{j_1} + x_{j_2} + \cdots + x_{j_k}$
- $f_{RSP} = \prod_{i=1}^{n} f_{RS}^{j}$
- $f_{Parity} = (-1)^{|S|}$
- $f_{Ryser} = f_{Parity} \cdot f_{RSP}$

Representing Ryser's Formula

$$perm(A) = (-1)^n \sum_{S \subseteq \{1,2,\ldots n\}} (-1)^{|S|} \prod_{i=1}^{n} \sum_{j \in S} a_{i,j}$$

- Boolean variable x_j for each column j
- $f_{RS}^{j} = x_{j_1} + x_{j_2} + \cdots + x_{j_k}$
- $f_{RSP} = \prod_{i=1}^{n} f_{RS}^{j}$
- $f_{Parity} = (-1)^{|S|}$
- $f_{Ryser} = f_{Parity} \cdot f_{RSP}$
- $perm(A) = (-1)^n \cdot \exists x_1, x_2 \ldots x_n \, f_{Ryser}$

Representing Rysers Formula

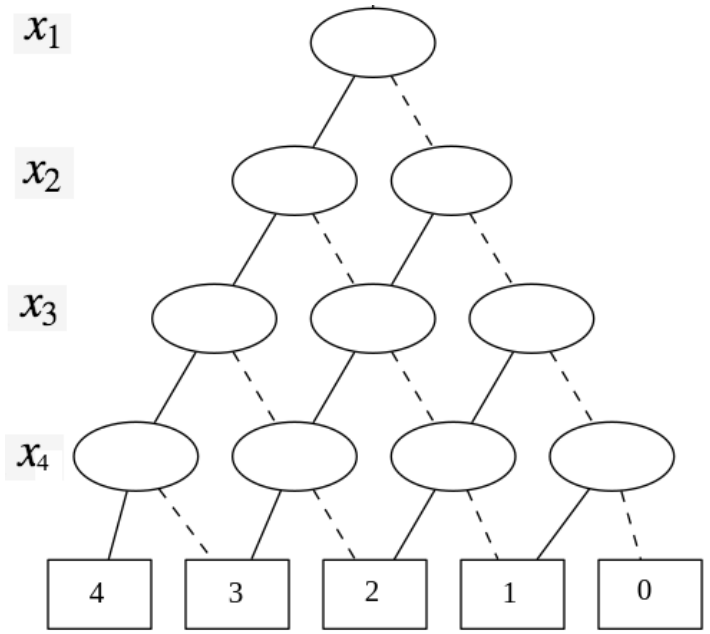
- Ex: 4×4 matrix of all 1s

- Observations:
  - $f_{RS}^{j} = \mathrm{x}_1 + \mathrm{x}_2 + \mathrm{x}_3 + \mathrm{x}_4$ for all j
  - $f_{RSP} = \left(f_{RS}^{j}\right)^4$

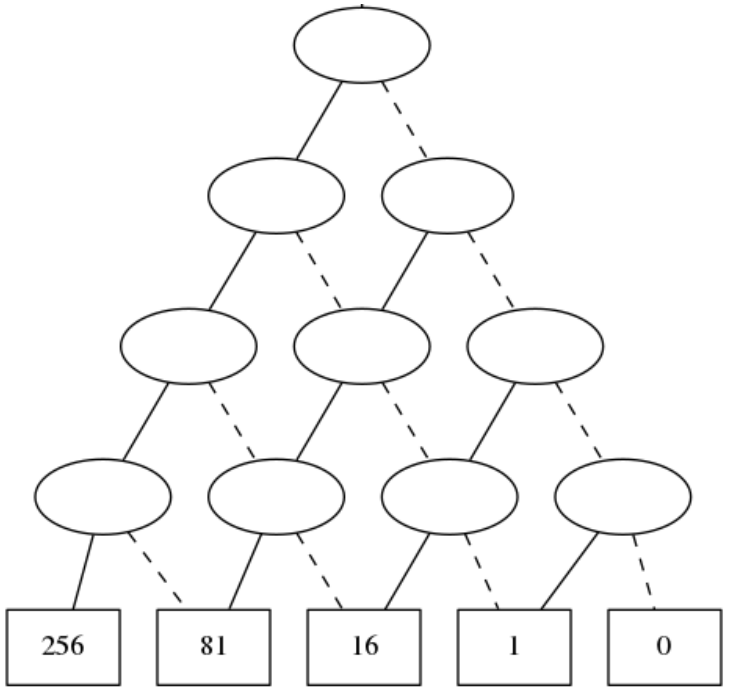
X_1	X_2	X_3	X_4
1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

Representing Rysers Formula

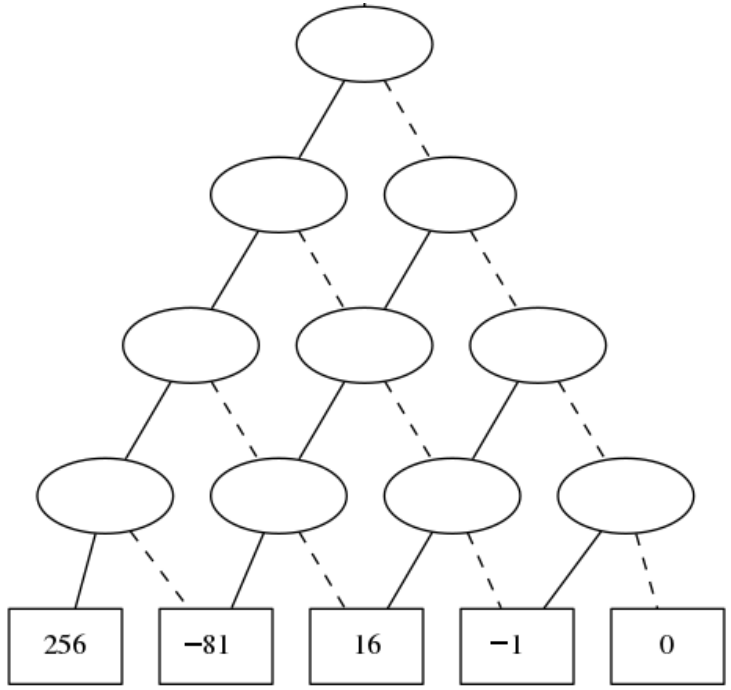
- Ex: 4×4 matrix of all 1s



Row-Sum



Row-Sum Product



Rysers

Early Abstraction

$$perm(A) = (-1)^n \cdot \exists x_1, .. x_n \, f_{Rysers}$$

Early Abstraction

$$perm(A) = (-1)^n \cdot \exists x_1, \ldots x_n \, f_{Parity} \cdot \prod_{i=1}^{n} f_{RS}^{i}$$

Early Abstraction

$$perm(A) = (-1)^n \cdot \exists x_1, \ldots x_n \, f_{Parity} \cdot \prod_{i=1}^{n} f_{RS}^{i}$$

- Push existential quantifiers inside the product
  - Not all $x_i s$ appear in each f_{RS}
- Keeps size of ADD small
- Many variable ordering heuristics in literature

Early Abstraction

$$perm(A) = (-1)^n \cdot (\exists x_j f_{RS}^{i_4} \ \ldots \left(\exists x_k f_{RS}^{i_3} \cdot \left(\exists x_1, x_5, x_6 \, f_{Parity} \cdot f_{RS}^{i_1} \cdot f_{RS}^{i_2}\right)\right)))$$

- Push existential quantifiers inside the product
  - Not all $x_i s$ appear in each f_{RS}
- Keeps size of ADD small
- Many variable ordering heuristics in literature

Implementation

- Sylvan library [van Dijk,'15] implements ADDs with arbitrary precision arithmetic
- Natively supports parallelization
- Our tool is called *RysersADD*
  - *RysersADD-P with* no additional effort

Experimental Setup

- Algorithm Suite
  - RysersADD, RysersADD-P (12 cores)
  - Model Counters: D4, DSharp
    - Encode Perfect-Matching (ExactOne Constraints) into CNF
    - 6 encodings: Pairwise, Sequential, Bitwise, Ladder, M/I Totalizer
  - Other: Explicit Ryser's Algorithm, CountAtom, ADDMC, C2D, B+E

Experimental Setup

- Benchmarks
  - Random: **Dense**
    - Start with a matrix of identical rows

$$\begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix}$$

Experimental Setup

- Benchmarks
  - Random: **Dense**
    - Start with a matrix of identical rows
    - Flip $C_f \cdot n$ entries uniformly at random

$$\begin{vmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{vmatrix}$$

Experimental Setup

- Benchmarks
  - Random: **Sparse**
    - Start with a matrix of identical rows

$$\begin{vmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$

Experimental Setup

- Benchmarks
  - Random: **Sparse**
    - Start with a matrix of identical rows
    - Flip $C_f \cdot n$ entries uniformly at random

$$\begin{vmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{vmatrix}$$

Experimental Setup

- Benchmarks
  - Random: **Other Similar Rows**
    - Start with a matrix of identical rows

$$\begin{vmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{vmatrix}$$

Experimental Setup

- Benchmarks
  - Random: **Other Similar Rows**
    - Start with a matrix of identical rows
    - Flip $C_f \cdot n$ entries uniformly at random

$$\begin{vmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{vmatrix}$$

Experimental Setup

- Benchmarks
  - Random: **Dense**, Sparse, Other Similar rows
    - Start with a matrix of identical rows
    - Flip $C_f \cdot n$ entries uniformly at random

Table 1: Parameters used for generating random matrices

Experiment	**Matrix Size n**	C_f**, where** $C_f \cdot n$ **matrix entries flipped**	**Starting Matrix Row Density** ρ	**#Instances**	**Total Benchmarks**
Dense	30, 40, 50, 60, 70	1, 1.1, 1.2, 1.3, 1.4	1	20	500
Sparse	30, 40, 50, 60, 70	3.9, 4.3, 4.7, 5.1, 5.5	0	20	500
Similar	40, 50, 60, 70, 80	1, 1.1, 1.2, 1.3, 1.4	0.7, 0.8, 0.9	15	1125

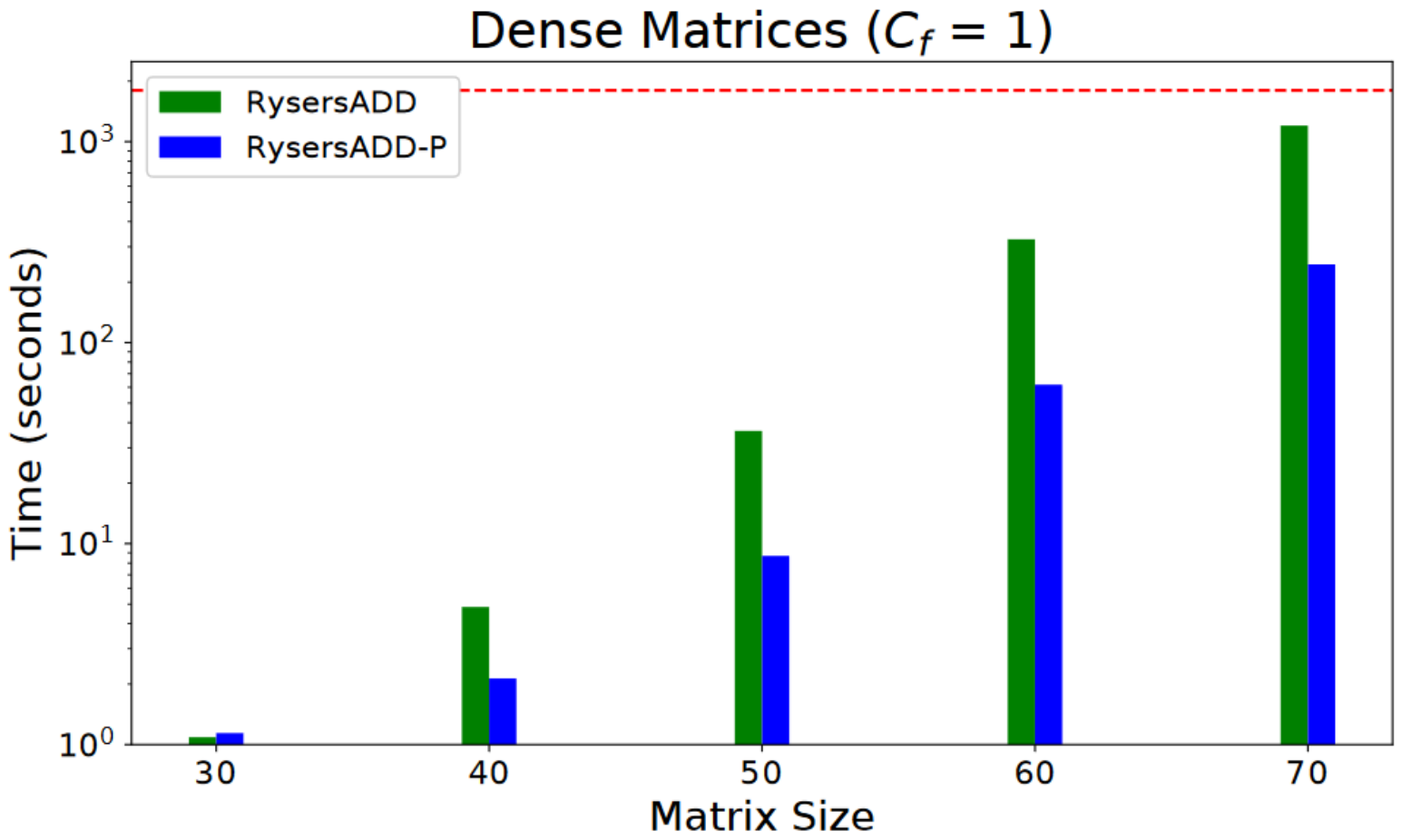
Experimental Setup

- Benchmarks
  - Random: **Dense**, Sparse, **Other Similar rows**
  - **SuiteSparse Matrix Collection**
  - Fullernes C_{60}, C_{100}

Observations

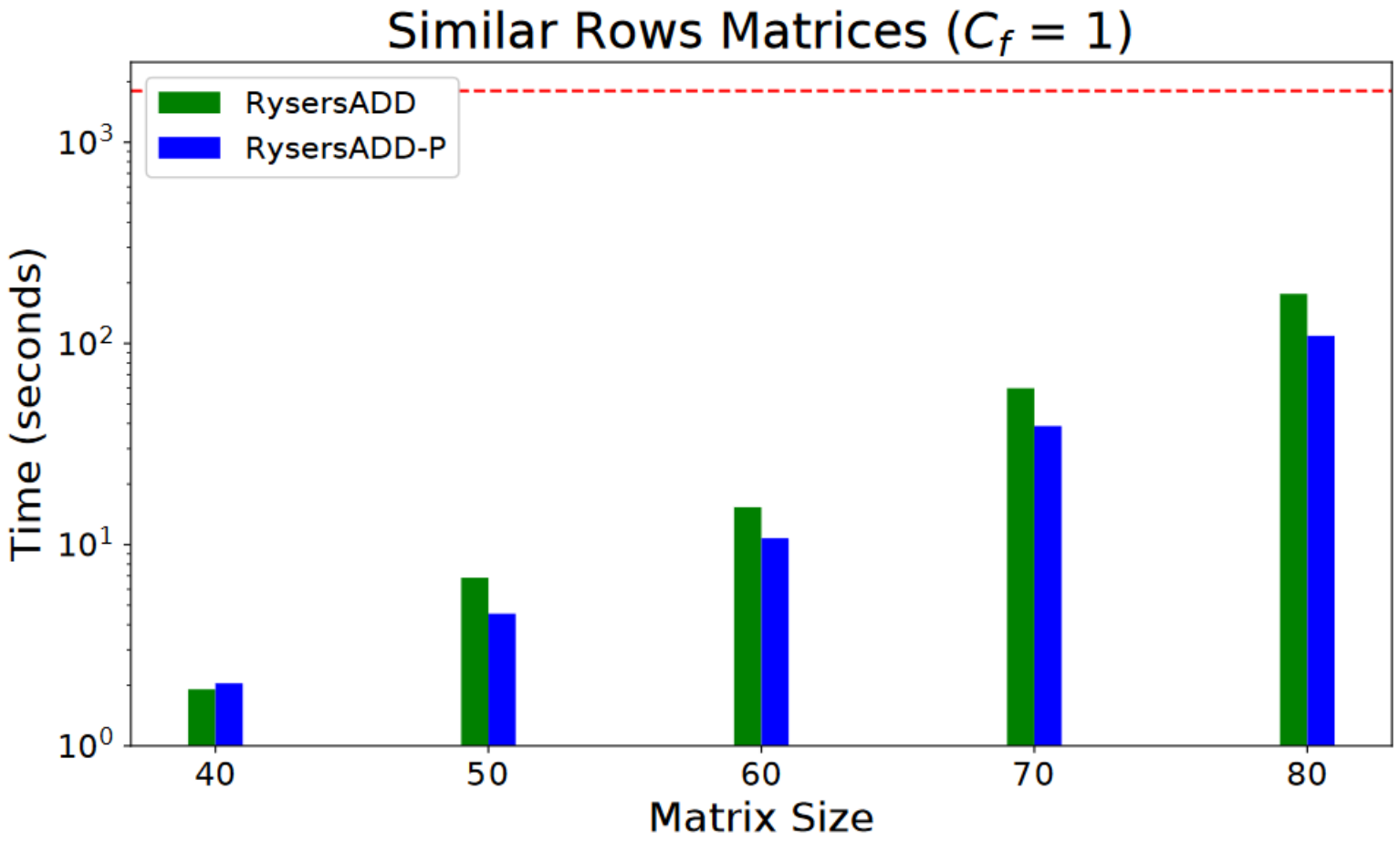
- ADD-based methods vastly outperform other approaches for non-sparse matrices
- D4 is the best performer for sparse matrices
  - Performance of RysersADD gets better as sparsity decreases

Results: Dense Matrices



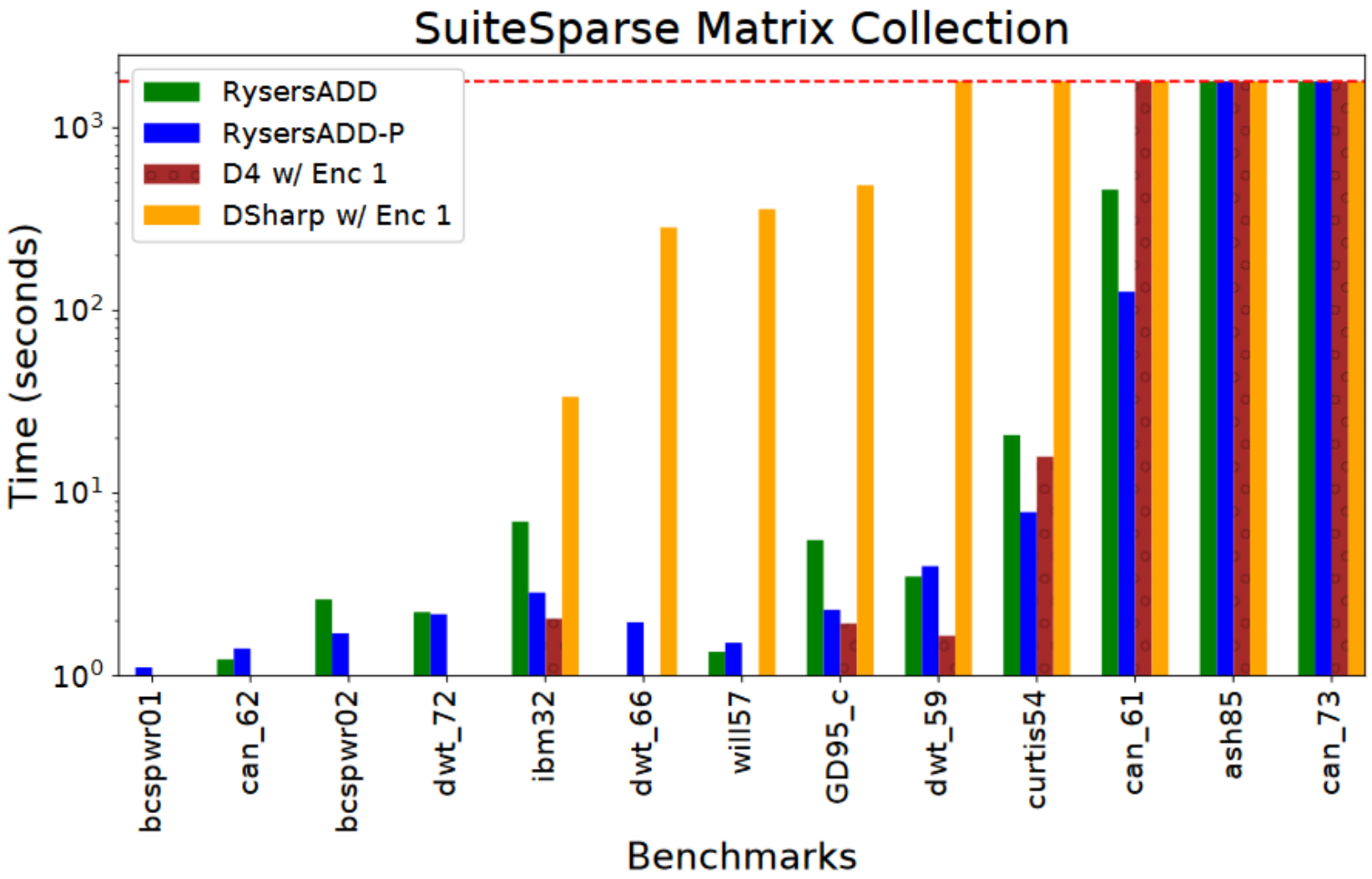
D4, Dsharp (not shown) timeout on all instances

Results: Other "Similar-Rows" Matrices



D4, Dsharp (not shown) timeout on all instances

Results: SuiteSparse Collection

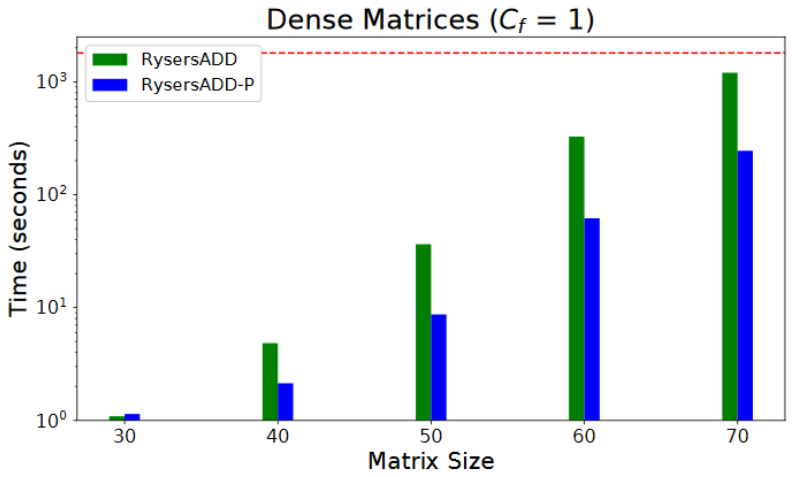


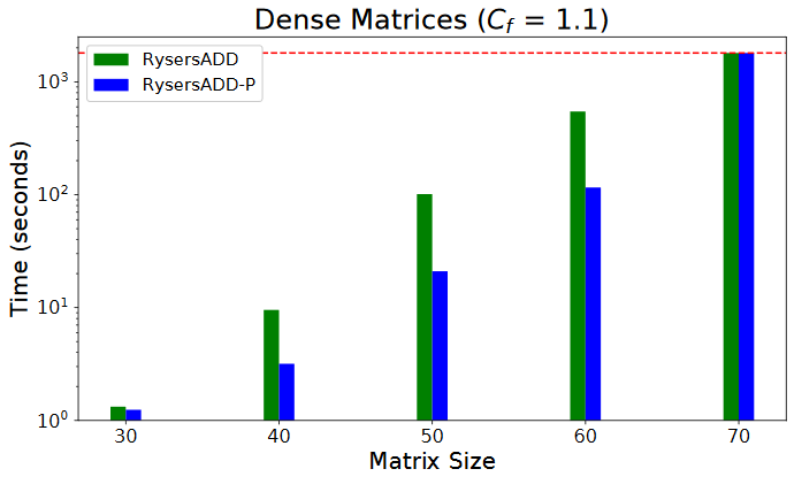
Takeaways

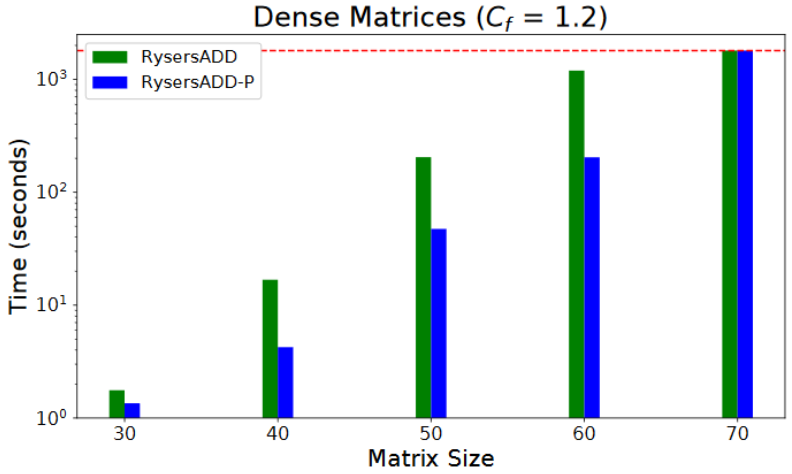
- Symbolic approaches show promise
  - More applications of ADDs?
- Define a new tractable subclass of matrices?
- Better bounds for dense matrices?
- Open Question: Exponentially faster algorithm than Ryser's?

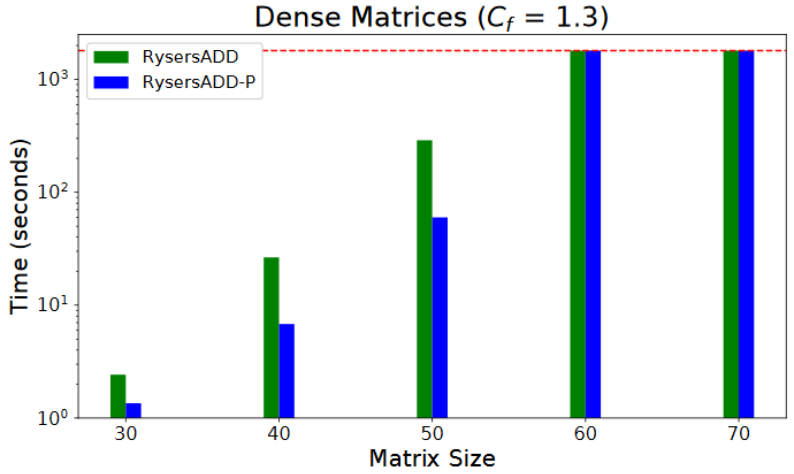
Backup

Results: Dense Matrices

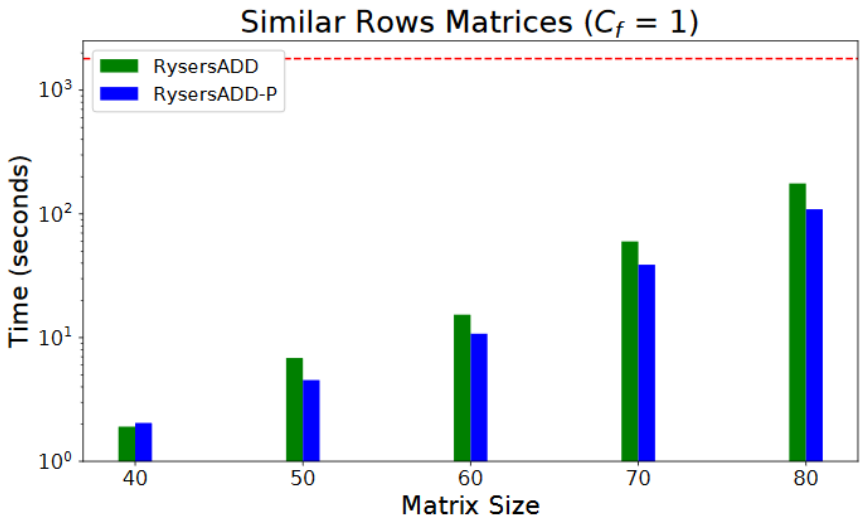


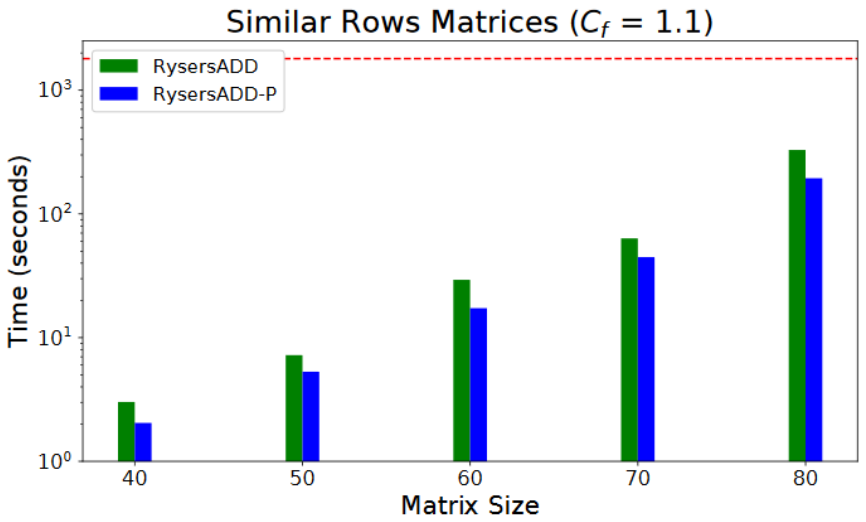


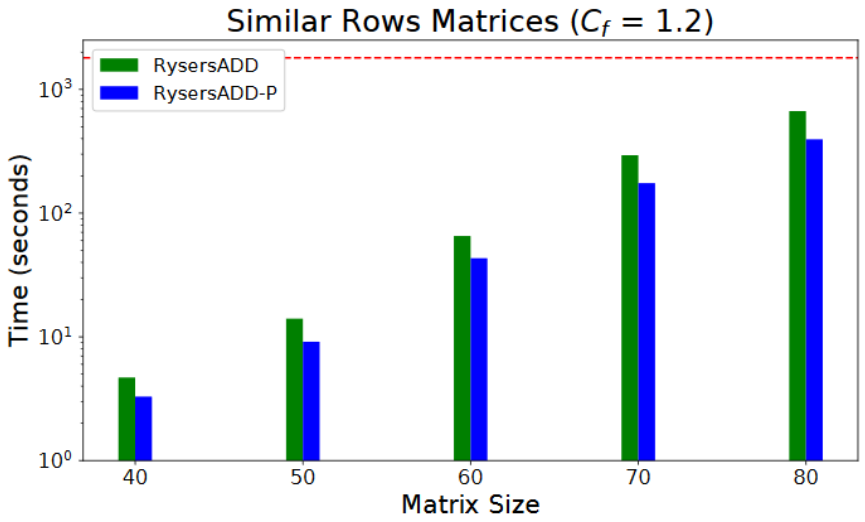


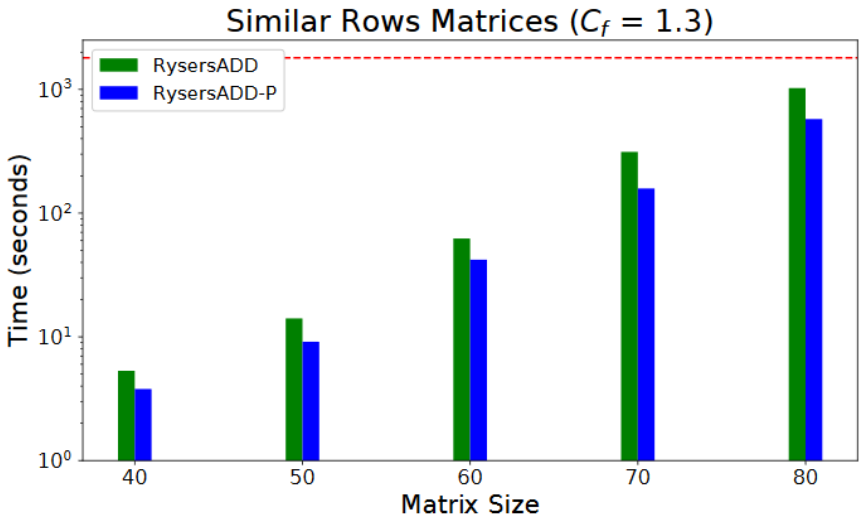


Results: Other Similar Rows Matrices

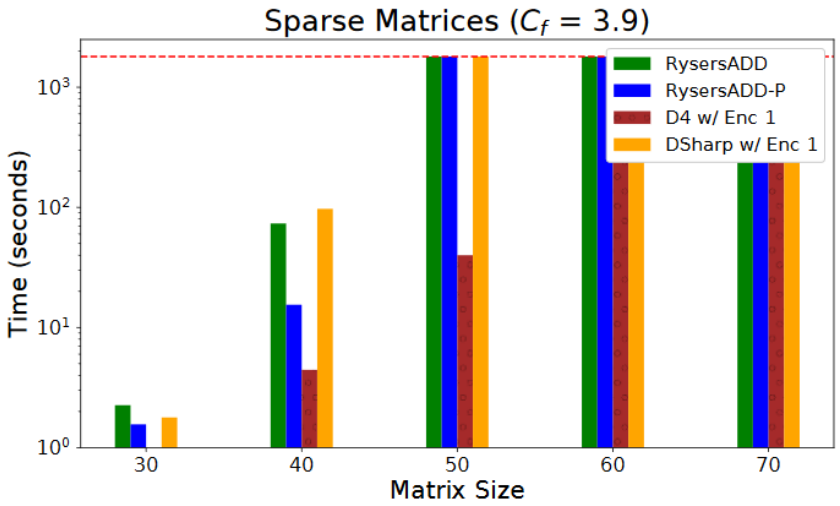


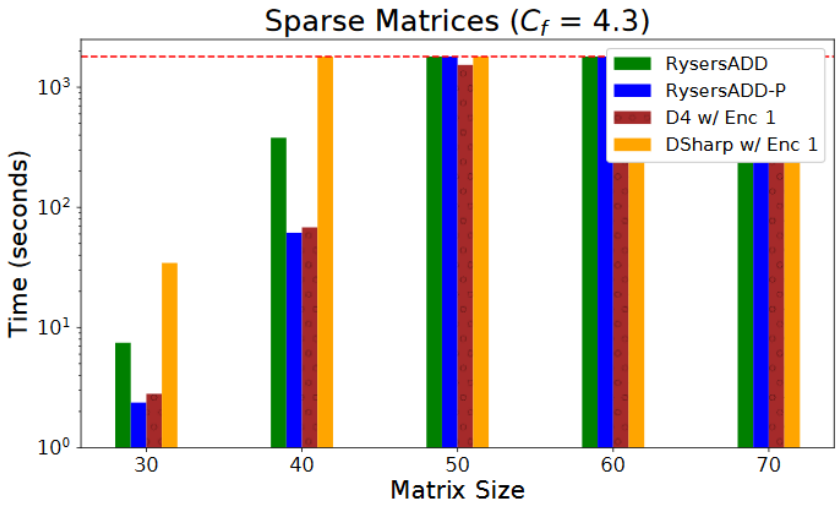


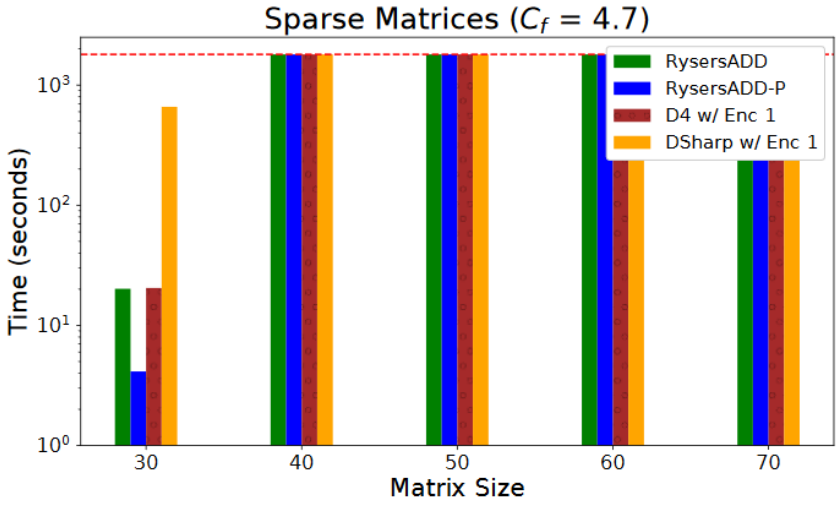


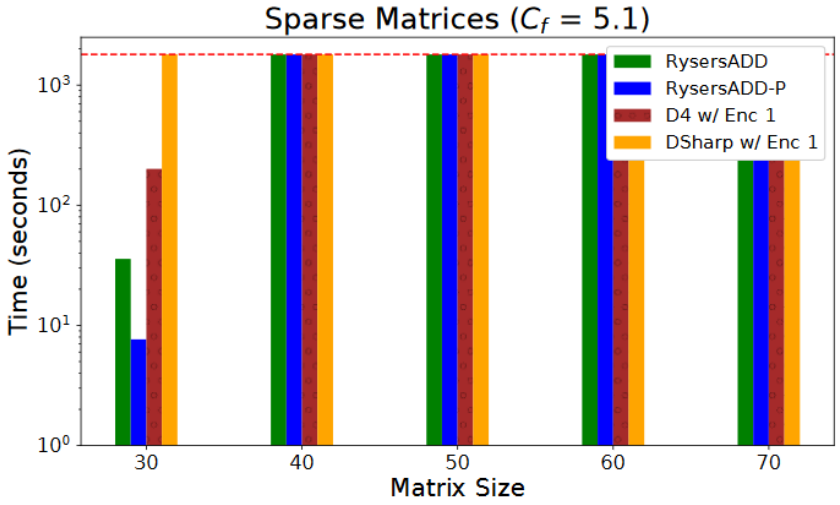


Results: Sparse Matrices







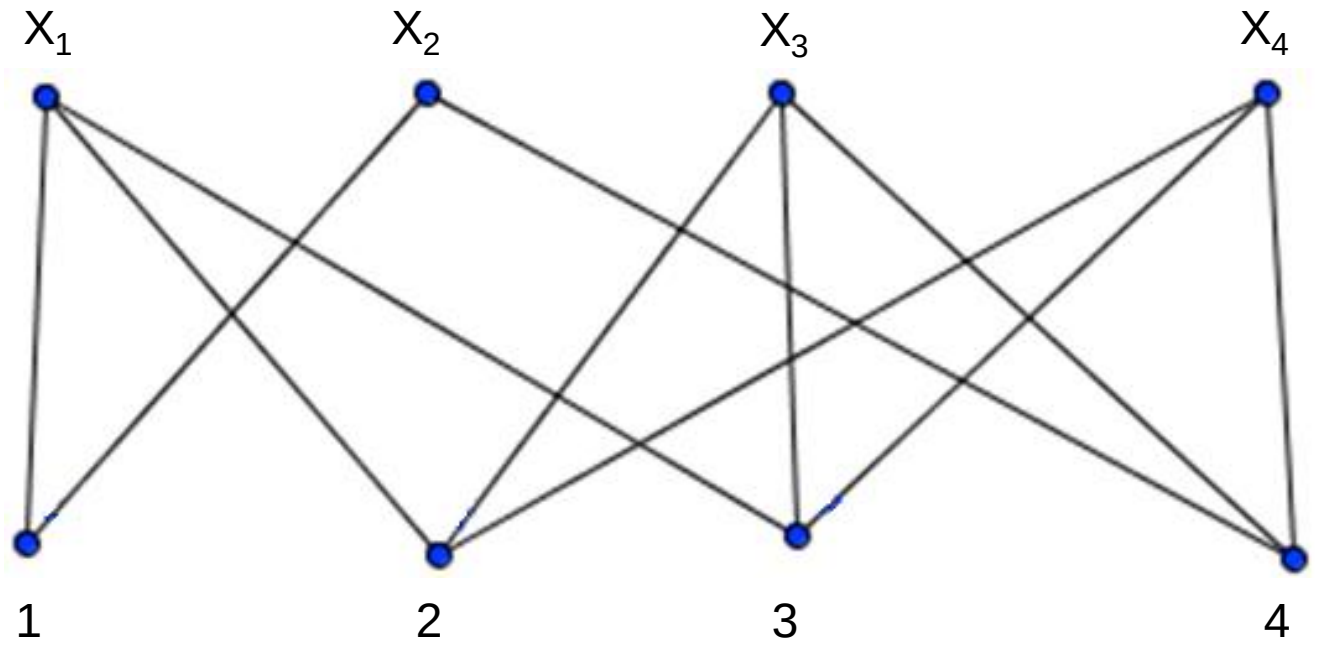


Results: Adjacency Matrices of Fullerenes

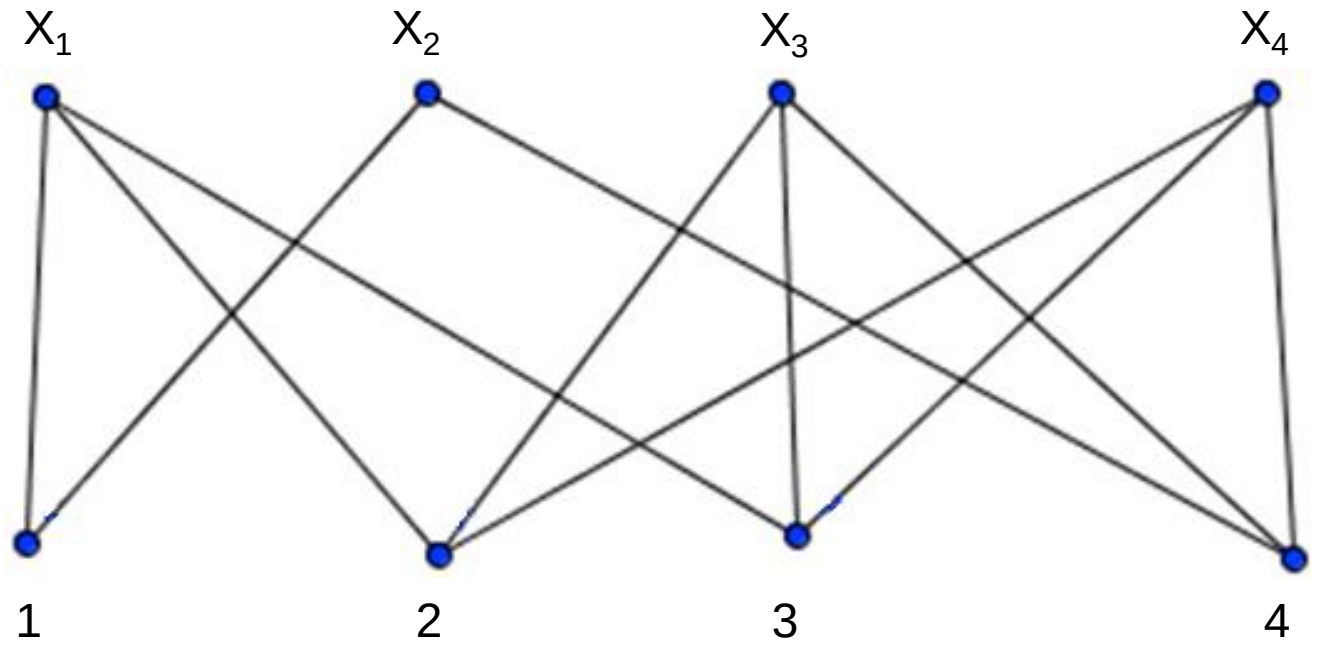
- C_{60}
  - D4: 94 sec
  - RysersADD: 96 sec
  - RysersADD-P: 57 sec
  - [Liang et al., ’04]*: 355 sec
  - [Liang et al., ‘13]*: 5 sec
- C_{100}
  - D4, RysersADD, RysersADD-P: Timeout
  - [Liang et al., ’13]*: 836 sec

*Not on same compute platform

All-Different Constraint



All-Different Constraint

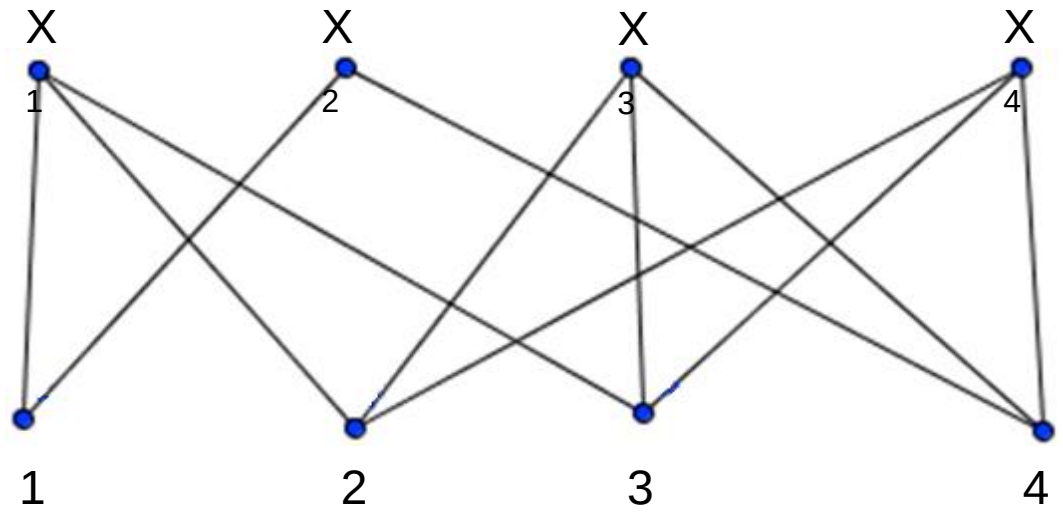


Number of valid assignments?

0/1 Permanent and Bipartite Perfect Matchings

- Sufficient to consider 0/1 matrices
- Can be interpreted as a bi-adjacency matrix
- Each 1-permutation is a perfect matching

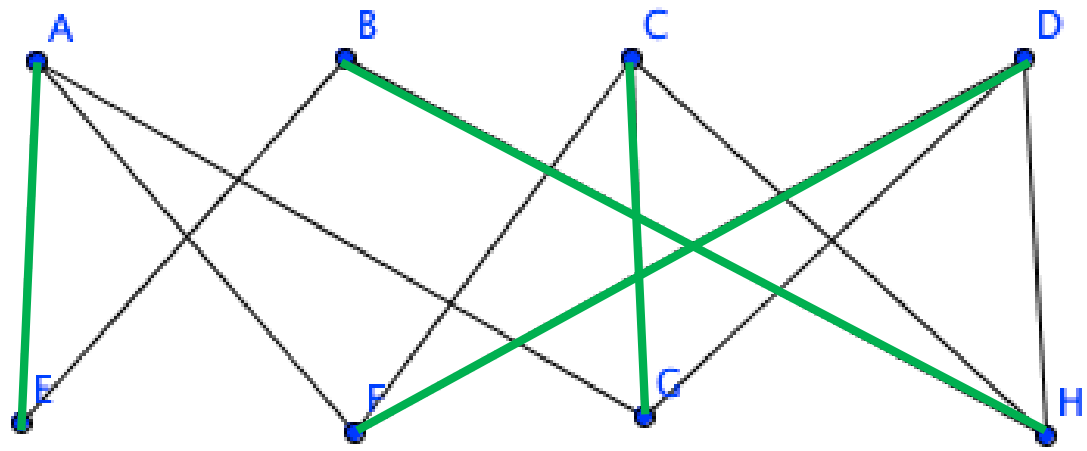
	E	F	G	H
A	1	1	1	0
B	1	0	0	1
C	0	1	1	1
D	0	1	1	1



0/1 Permanent and Bipartite Perfect Matchings

- Sufficient to consider 0/1 matrices
- Can be interpreted as a bi-adjacency matrix
- Each 1-permutation is a perfect matching

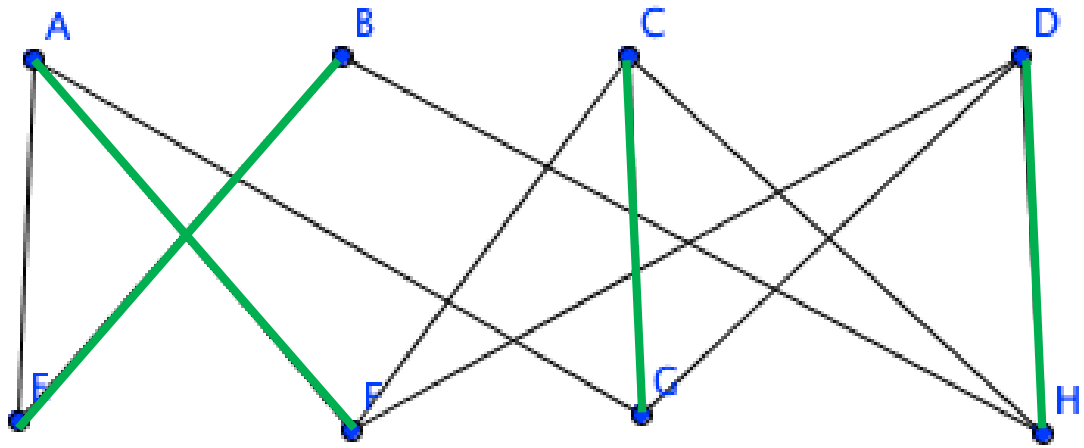
	E	F	G	H
A	1	1	1	0
B	1	0	0	1
C	0	1	1	1
D	0	1	1	1



0/1 Permanent and Bipartite Perfect Matchings

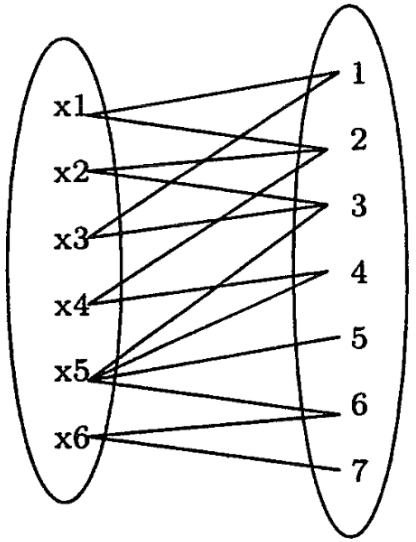
- Sufficient to consider 0/1 matrices
- Can be interpreted as a bi-adjacency matrix
- Each 1-permutation is a perfect matching

	E	F	G	H
A	1	1	1	0
B	1	0	0	1
C	0	1	1	1
D	0	1	1	1



Applications

- Chemistry: Stability of Fullerenes
- Solving CSPs: Counting-Based Heuristics for AllDifferent Constraint [Pesant et al., ’12]



- Quantum Computing: To see how far classical computation can be pushed [Bjorklund et al., ‘18]