Chordal Sparse Matrices and Semidefinite Programming

Theo Diamandis
tdiamand@mit.edu

Semidefinite Programs (SDPs) have PSD constraints

Primal Problem: variable $Z \in \mathbf{S}^n$

$$\begin{array}{ll} \text{maximize} & \mathbf{Tr}\,(GZ) \\ \text{s.t.} & \mathbf{Tr}(F_i Z) + c_i = 0, \quad i = 1, \ldots m \\ & Z \succeq 0, \end{array}$$

Dual problem: variable $x \in \mathbf{R}^m$

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{s.t.} & -\sum_i x_i F_i - G \succeq 0 \end{array}$$

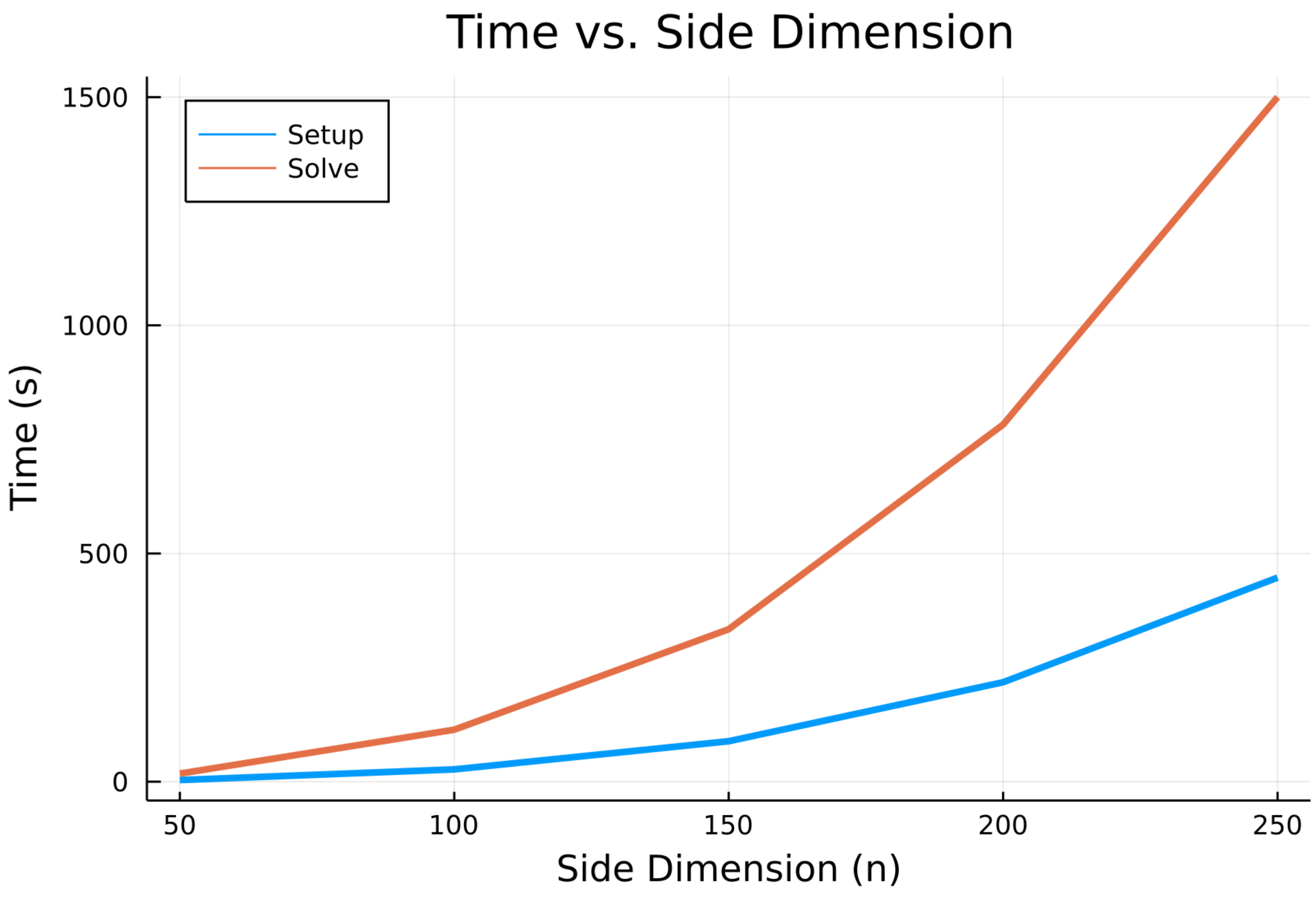
SDPs are a powerful modeling tool

- Signal Proc & Control
  - Phase Retrieval (e.g. imaging)
  - Polynomial controller design
- Stats/ML
  - Outlier Detection (e.g. ellipsoidal peeling)
  - Experiment Design
  - Factor Analysis
  - Low-rank matrix completion & decomp.
  - Neural network robustness verification

- Circuit Design
- Portfolio Optimization
- Combinatorial Optimization
- Mechanical structure optimization
- Operations Research
  - Facility Location
  - Scheduling
- Robust optimization
- Geometric data processing & CV

SDPs are plagued by scaling issues

Per iteration cost is (approx) cubic in the side dimension



But we can exploit sparsity

Variable $Z \in \mathbf{S}^n$

Only use, for example, 2% of entries of Z

$$\left.\begin{array}{ll} \text{maximize} & \mathbf{Tr}\,(GZ) \\ \text{s.t.} & \mathbf{Tr}(F_i Z) + c_i = 0, \quad i = 1, \ldots m \end{array}\right\} \textbf{Sparse}$$

$$\qquad\qquad Z \succeq 0,$$

Goal: get rid of the "dense" $n \times n$ PSD constraint $\to$ reduce # variables by 98%

Need: a check that a sparse matrix can be "completed" to be a PSD matrix

Chordal Decomposition: check $Z \succeq 0$ via sub matrices

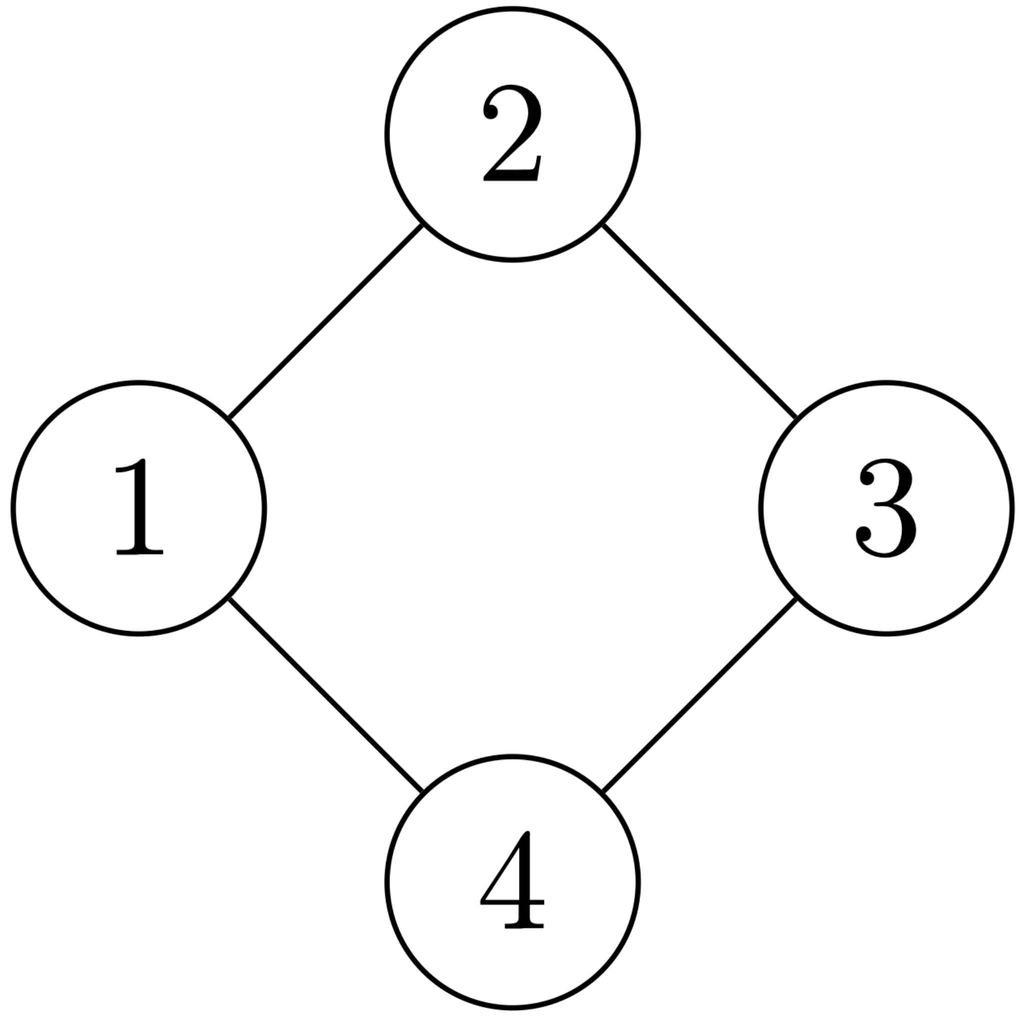
$$\begin{array}{cc} & \begin{matrix}1 & 2 & 3 & 4 & 5\end{matrix} \\ \begin{matrix}1\\2\\3\\4\\5\end{matrix} & \begin{bmatrix} \bullet & \bullet & \bullet & & \\ \bullet & \bullet & \bullet & & \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ & & \bullet & \bullet & \bullet \\ & & \bullet & \bullet & \bullet \end{bmatrix} \end{array} \succeq 0 \iff \begin{array}{cc} & \begin{matrix}1 & 2 & 3 & 4 & 5\end{matrix} \\ \begin{matrix}1\\2\\3\\4\\5\end{matrix} & \begin{bmatrix} \bullet & \bullet & \bullet & & \\ \bullet & \bullet & \bullet & & \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ & & \bullet & \bullet & \bullet \\ & & \bullet & \bullet & \bullet \end{bmatrix} \end{array} \begin{matrix} \succeq 0 \\ \succeq 0 \end{matrix}$$

$$O(5^3) = C \cdot 125 \qquad\qquad O(2 \cdot 3^3) = C \cdot 54$$

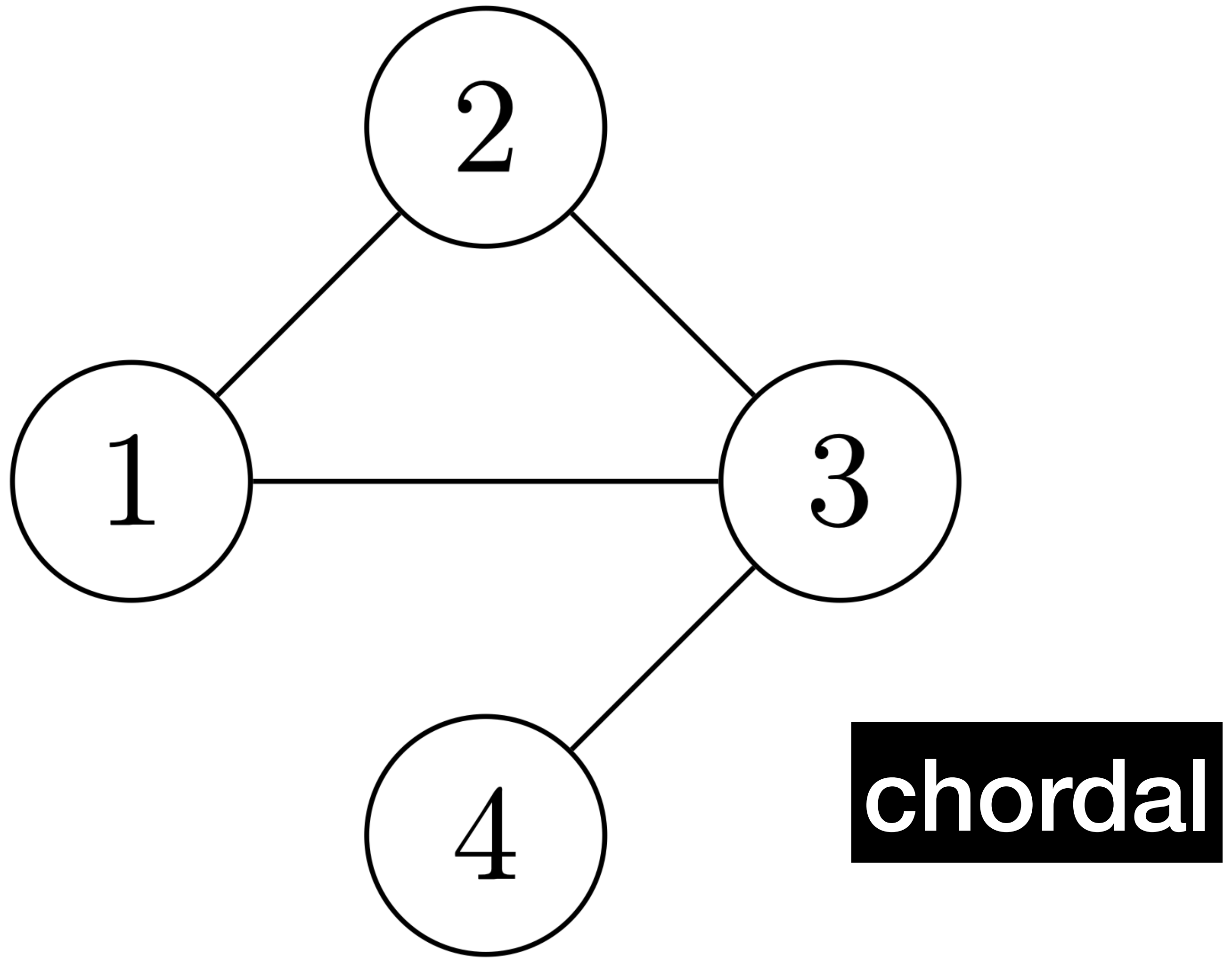
Detour: Chordal Graph Theory

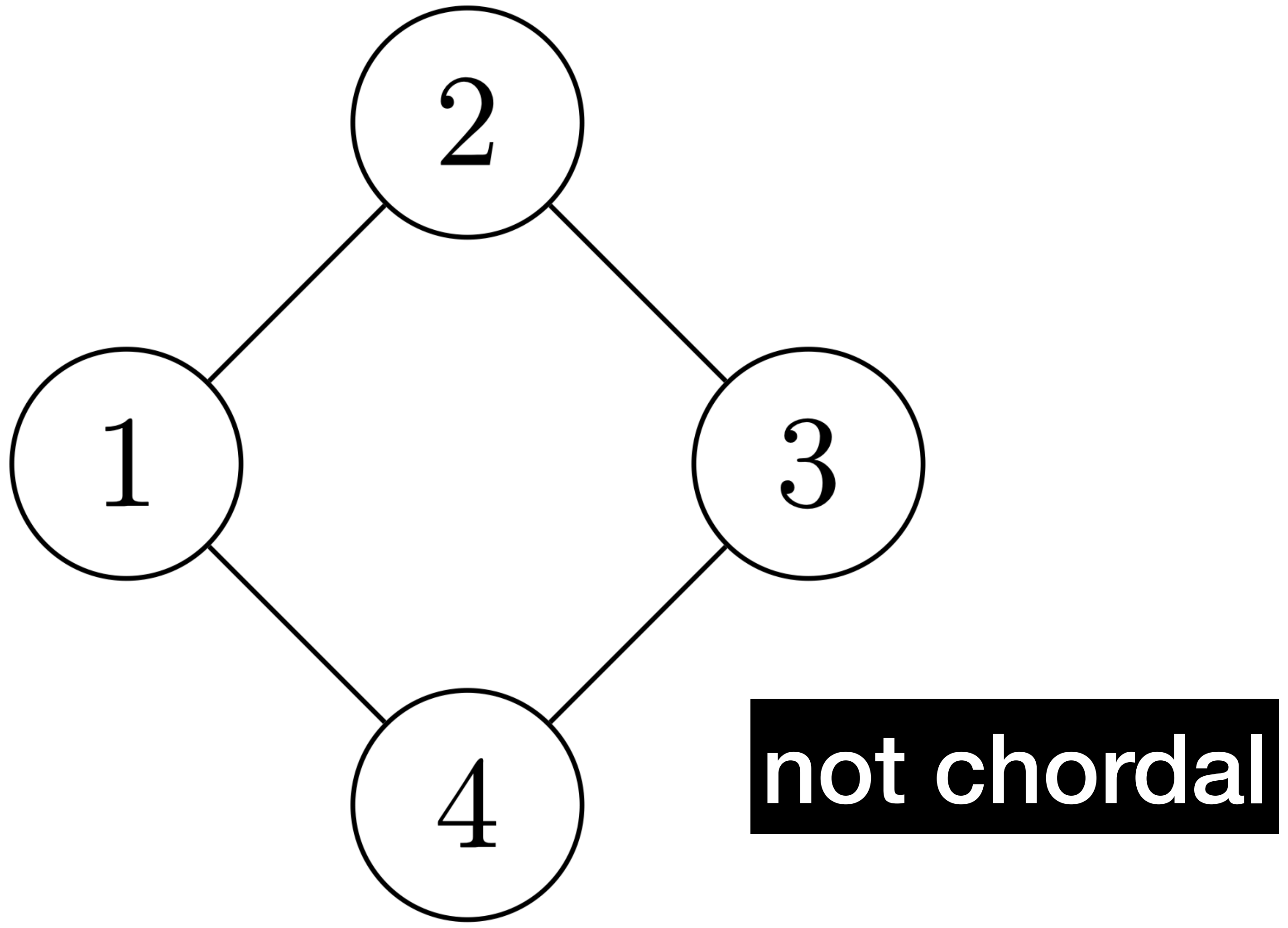
Sparsity patterns = undirected graphs

$$\begin{bmatrix} 11 & 21 & \mathbf{0} & 41 \\ 21 & 22 & 32 & \mathbf{0} \\ \mathbf{0} & 32 & 33 & 43 \\ 41 & \mathbf{0} & 43 & 44 \end{bmatrix} \Longrightarrow \begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{bmatrix} \bullet & \bullet & & \bullet \\ \bullet & \bullet & \bullet & \\ & \bullet & \bullet & \bullet \\ \bullet & & \bullet & \bullet \end{bmatrix} \end{array} \Longleftrightarrow$$



Sparsity patterns = undirected graphs





Definition: A graph is *chordal* if all cycles of length > 3 have a chord

- Any graph can be made chordal by adding edges
- Many "hard" graph problems are "easy" on chordal graphs

Definition: A *clique* is a set of vertices that induces a maximal complete subgraph

Examples (Code Demo)

Example sparsity patterns:

- Banded
- Arrow
- Random

- Think about which are Chordal

Check if Z is PSD by looking at sub matrices

[Grone et al. 1984]

Theorem: A sparse matrix Z **with a chordal sparsity pattern** can be PSD completed if and only if

$$Z_{C_p} \succeq 0, \qquad p = 1, \ldots, P$$

where Z_{C_p} is the sub matrix of Z indexed by the clique C_p.

Applied to SDPs, reduces PSD constraint size

Matrices F_i, G have a chordal aggregate sparsity pattern

$$
\begin{array}{ll}
\text{maximize} & \mathbf{Tr}\,(GZ) \\
\text{s.t.} & \mathbf{Tr}(F_i Z) + c_i = 0, \quad i = 1, \ldots m \\
 & \cancel{Z \succeq 0}, \quad \boxed{Z_{C_p} \succeq 0,}
\end{array}
$$

See COSMO.jl & Chordal.jl

Iteration cost is now $O\left(\Sigma_{p=1}^{P} |C_p|^3\right)$ vs $O(n^3)$

Use similar approach for the dual problem

[Agler et al. 1988]

Theorem: A sparse matrix S **with a chordal sparsity pattern** is PSD if and only if there exist matrices $S_p \succeq 0$ such that

$$S = \sum_{p=1}^{P} T_p S_p T_p^T$$

where T_p is a selector matrix:

$$[T_p]_{ij} = \begin{cases} 1 & C_p(i) = j, \\ 0 & \text{otherwise} \end{cases}$$

Use a similar approach for the dual problem

Matrices F_i, G have a chordal aggregate sparsity pattern

$$
\begin{array}{ll}
\text{minimize} & c^T x \\
\text{s.t.} & -\sum_i x_i F_i - G \cancel{\succeq 0}
\end{array}
$$

$$
\boxed{\begin{array}{l} = \sum_{p=1}^{P} T_p S_p T_p^T \\ S_p \succeq 0 \end{array}}
$$

Algorithmic Details

These constructions lead to two new cones:

Sparse PSD-completable cone (primal problem):

$$\Pi_E(S_+^n) = \{X \mid \exists Y \in S_+^n \text{ s.t. } X = \Pi_E(Y)\}$$

Sparse PSD cone (dual problem):

$$S_+^n \cap S_E^n = \{X \mid X \succeq 0 \text{ and } (i,j) \in E \implies X_{ij} = 0\}$$

- We want to compute projections and gradients/Hessians of barriers

Reformulating → faster FO & IP algorithms

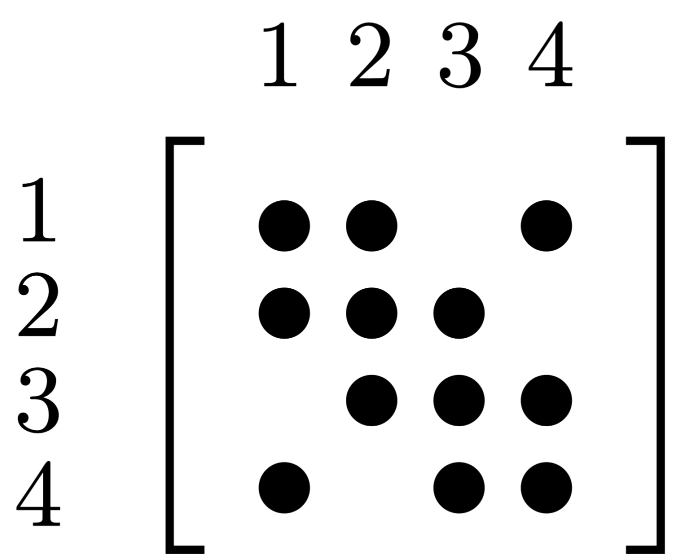
- For first order algorithms (e.g., ADMM), we project onto the PSD cone
  - Chordal decomposition → break up large PSD projection into P small ones

$$
\begin{aligned}
X^{k+1} &= \operatorname*{argmin}_X \left(f(X) + (\rho/2)\|X - Z^k + U^k\|_F^2 \right) \\
Z^{k+1} &= \Pi_{S_+^n} \left(X^{k+1} + U^k \right) \\
U^{k+1} &= U^k + X^{k+1} - Z^{k+1}.
\end{aligned}
$$

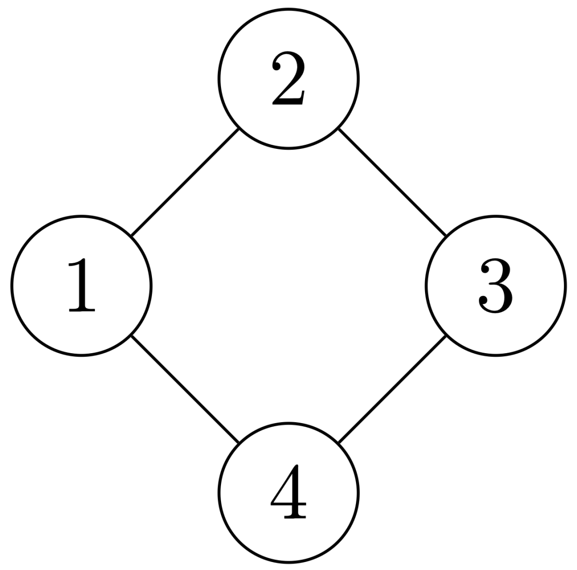
Reformulating → faster FO & IP algorithms

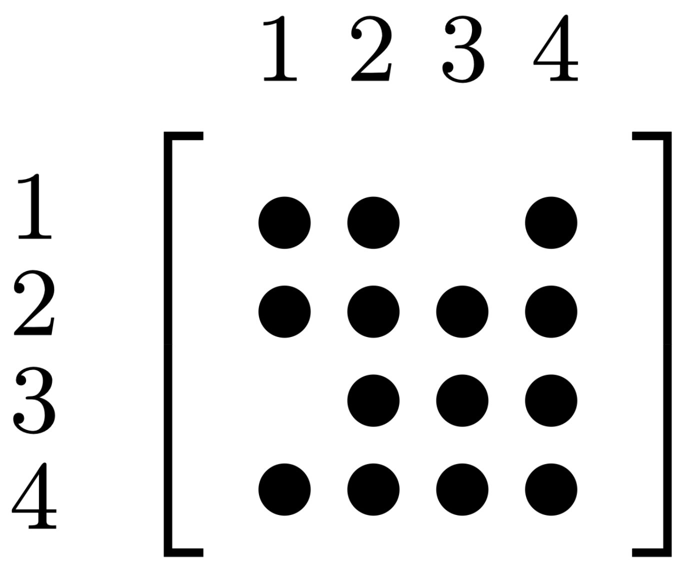
- For first order algorithms (e.g., ADMM), we project onto the PSD cone
  - Chordal decomposition → break up large PSD projection into P small ones
- For interior point algorithms, must compute gradient and/or Hessian of barrier
  - Chordal decomposition → efficiently compute these [ADV13, VA15]

Chordal Decomposition Steps



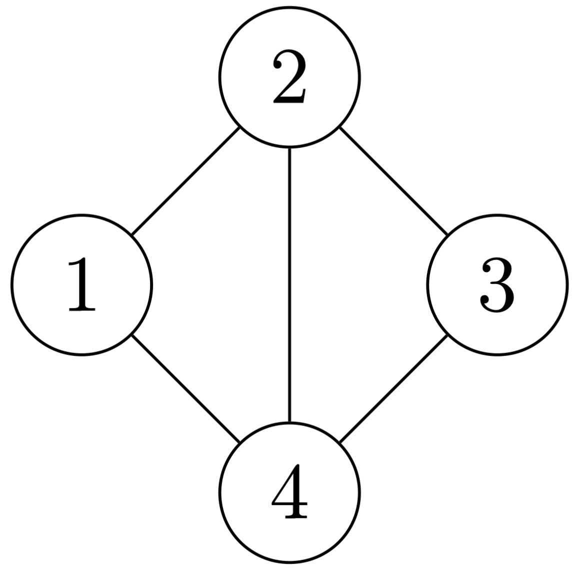
1 Input Sparsity Pattern

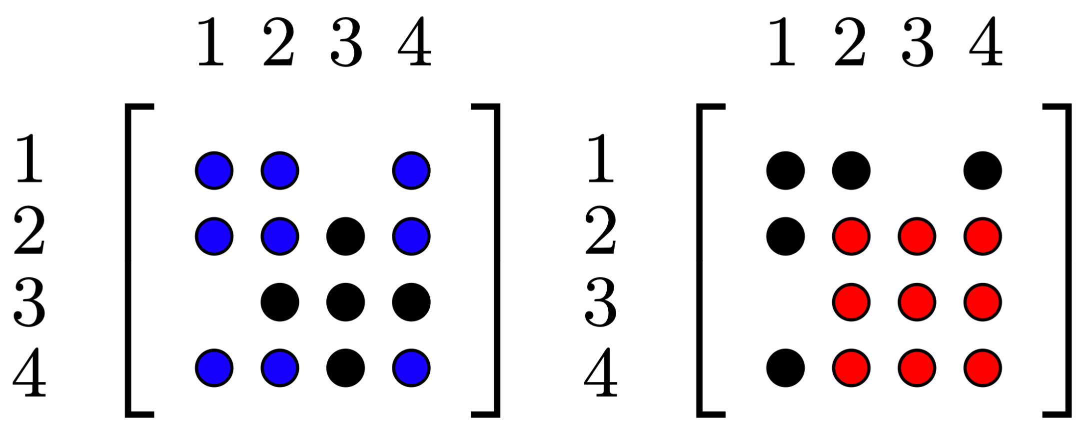




2 Chordal Extension

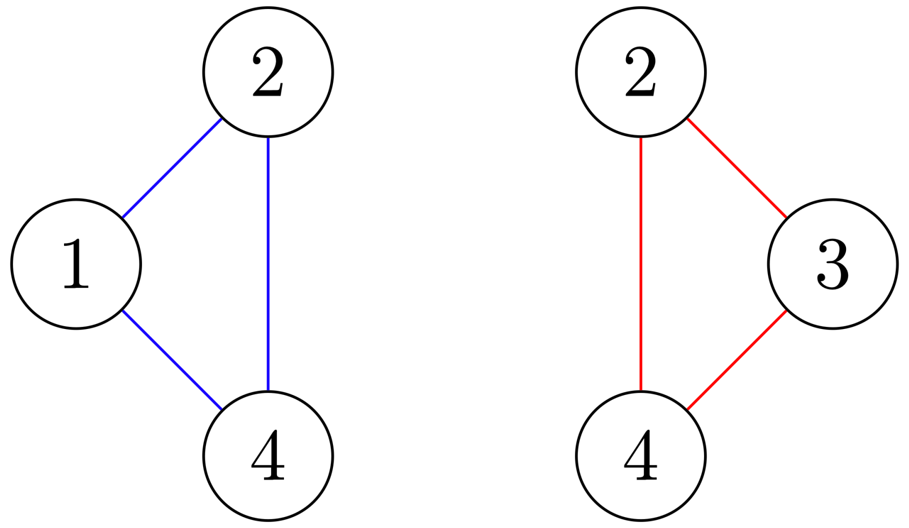
Reordering matters!





3 Find (& merge) cliques

[Michael Garstka et al. 2020]



4 Form SDP

5 Solve SDP

6 [primal] Reconstruct Z

We can reconstruct the full Z if needed

Our optimization variable Z is only partially specified

Two main approaches:

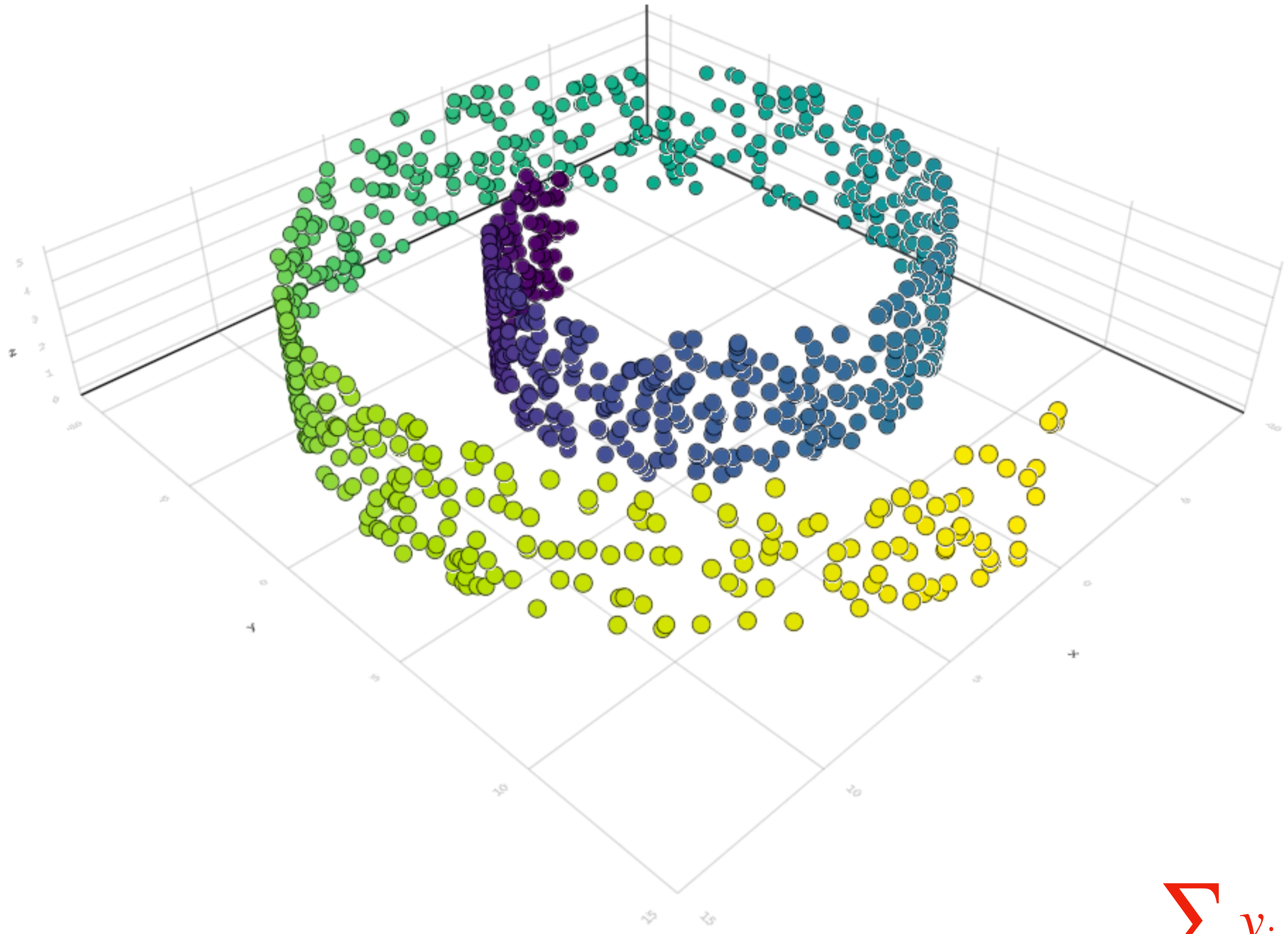
1. Maximum determinant completion [VA15]
    - Idea: Solve $ZL = L^{-T}D^{-1}$, where factorization is known from decomp
2. Minimum rank completion [Sun15]
    - Idea: factor each clique block & apply necessary orthogonal rotations

$$r = \max_p \mathbf{rank}(Z_{C_p})$$

Numerical Example

Example: Maximum Variance Unfolding

Preserve local distances + maximize (global) variance



Data: $x_i \in \mathbb{R}^{d_x}$ with pairwise distances $D_{ij} = \|x_i - x_j\|^2$

Goal: find embedding $y_i \in \mathbb{R}^{d_y}$ that preserves *local* distances

Method: solve an SDP with variable $Y = \tilde{Y}^T \tilde{Y}$, where

$$\tilde{Y} = \begin{bmatrix} | & & | \\ y_1 & \dots & y_n \\ | & & | \end{bmatrix} \in \mathbb{R}^{d_y \times n}$$

Variance

$$\begin{array}{lll} \text{maximize} & \mathbf{Tr}\, Y & \\ \text{s.t.} & Y_{ii} + Y_{jj} - 2Y_{ij} = D_{ij} & \forall (i,j) \in \mathcal{E}_k \\ & \mathbf{1}^T Y \mathbf{1} = 0 & \\ & Y \succeq 0. & \end{array}$$

$\sum y_i = 0$

Preserve kNN distances

MVU Sparsity Pattern

N = 250, k = 6

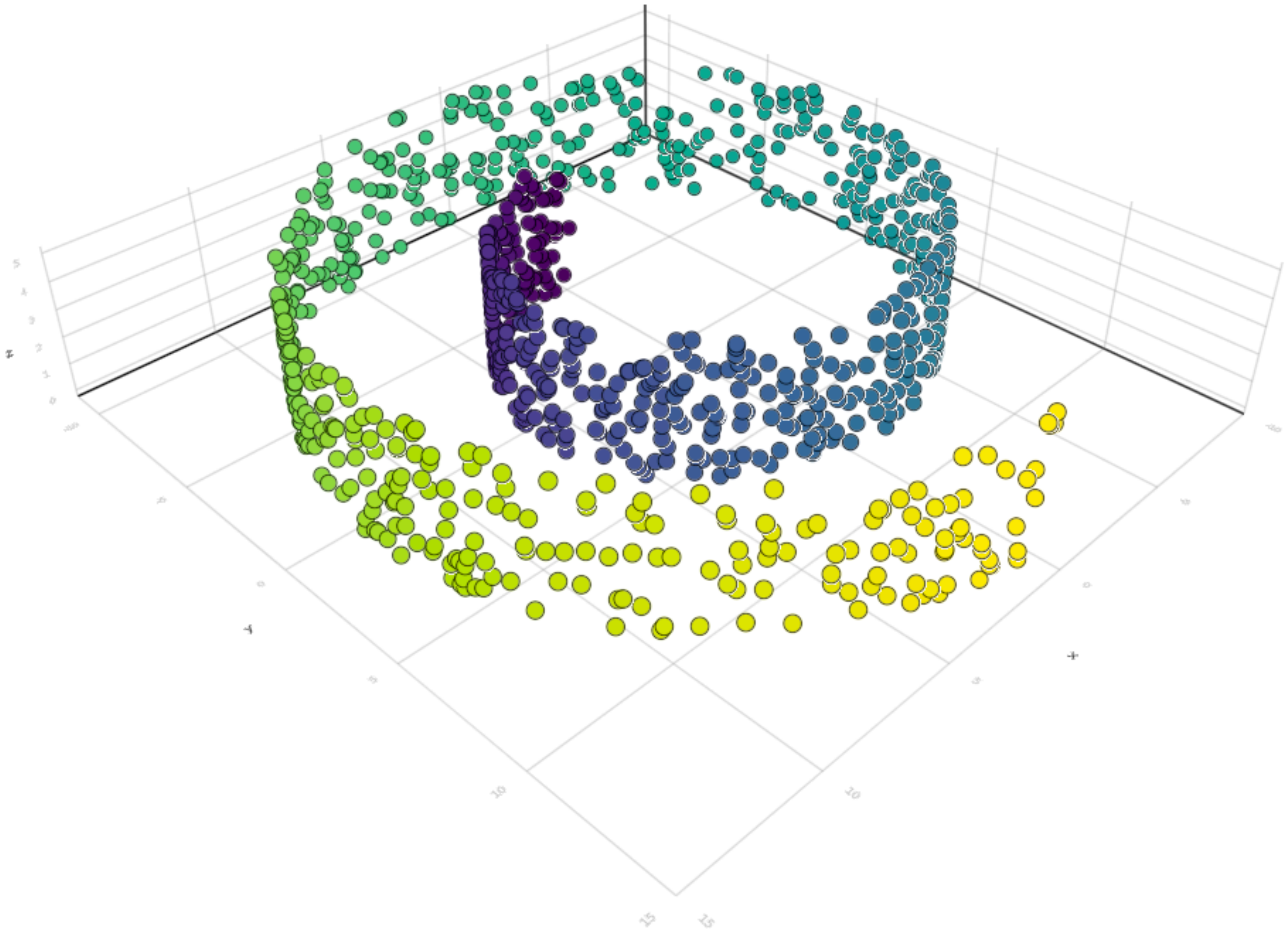
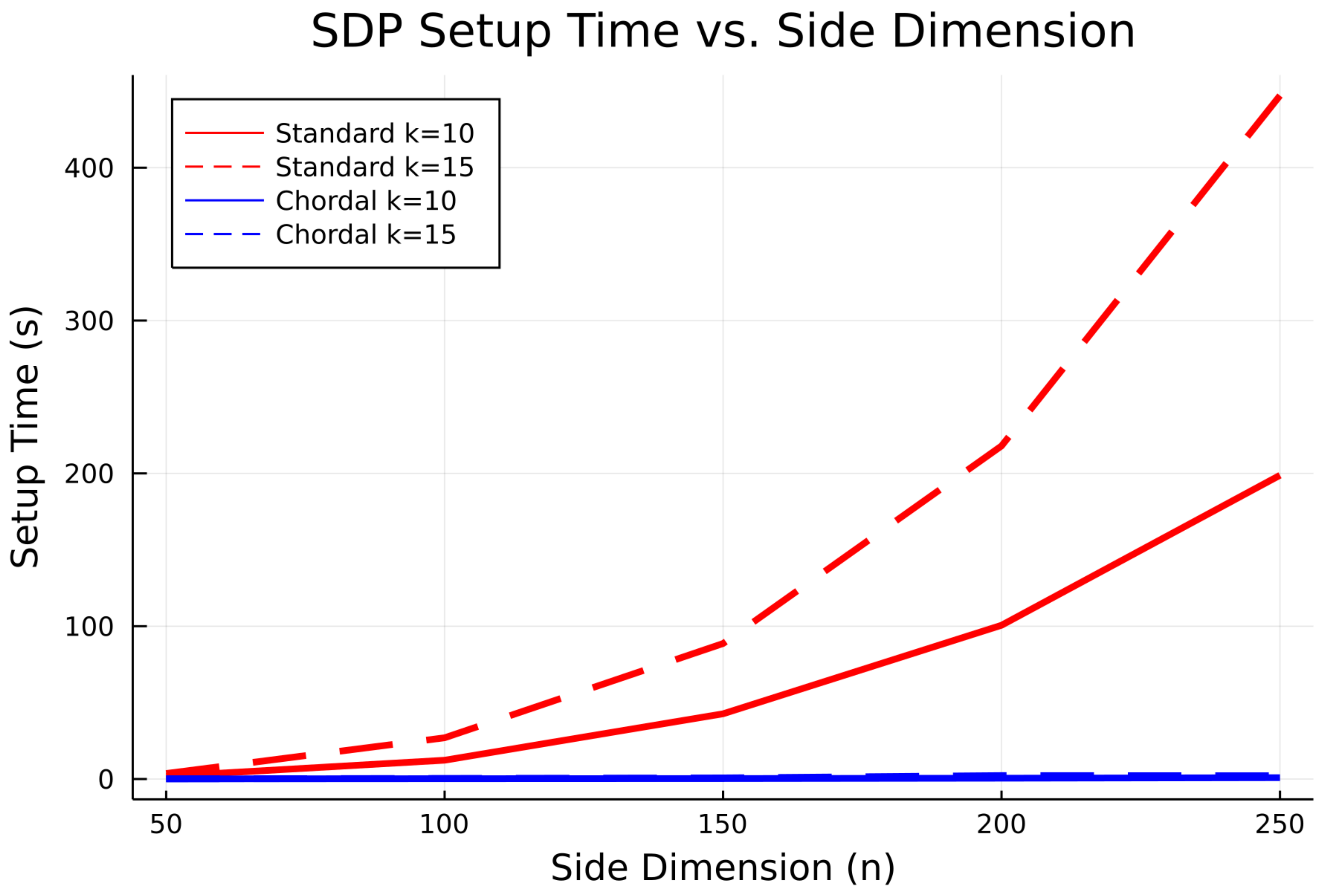
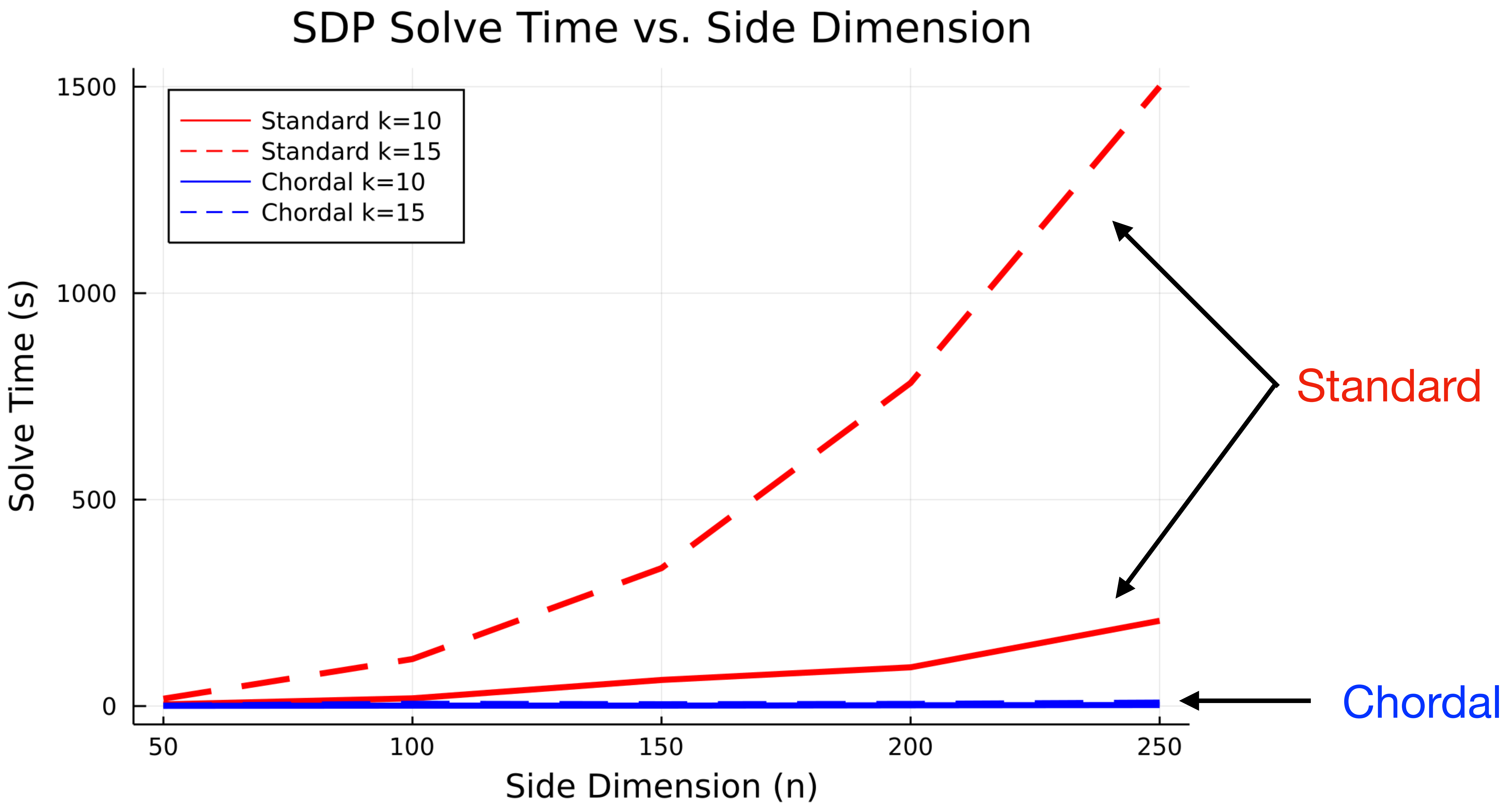


Figure 3: *Aggregate sparsity pattern for MVU with $n = 250$ and $k = 6$, which has 6487 nonzeros (left) and its chordal extension, which has 7382 nonzeros (right).*

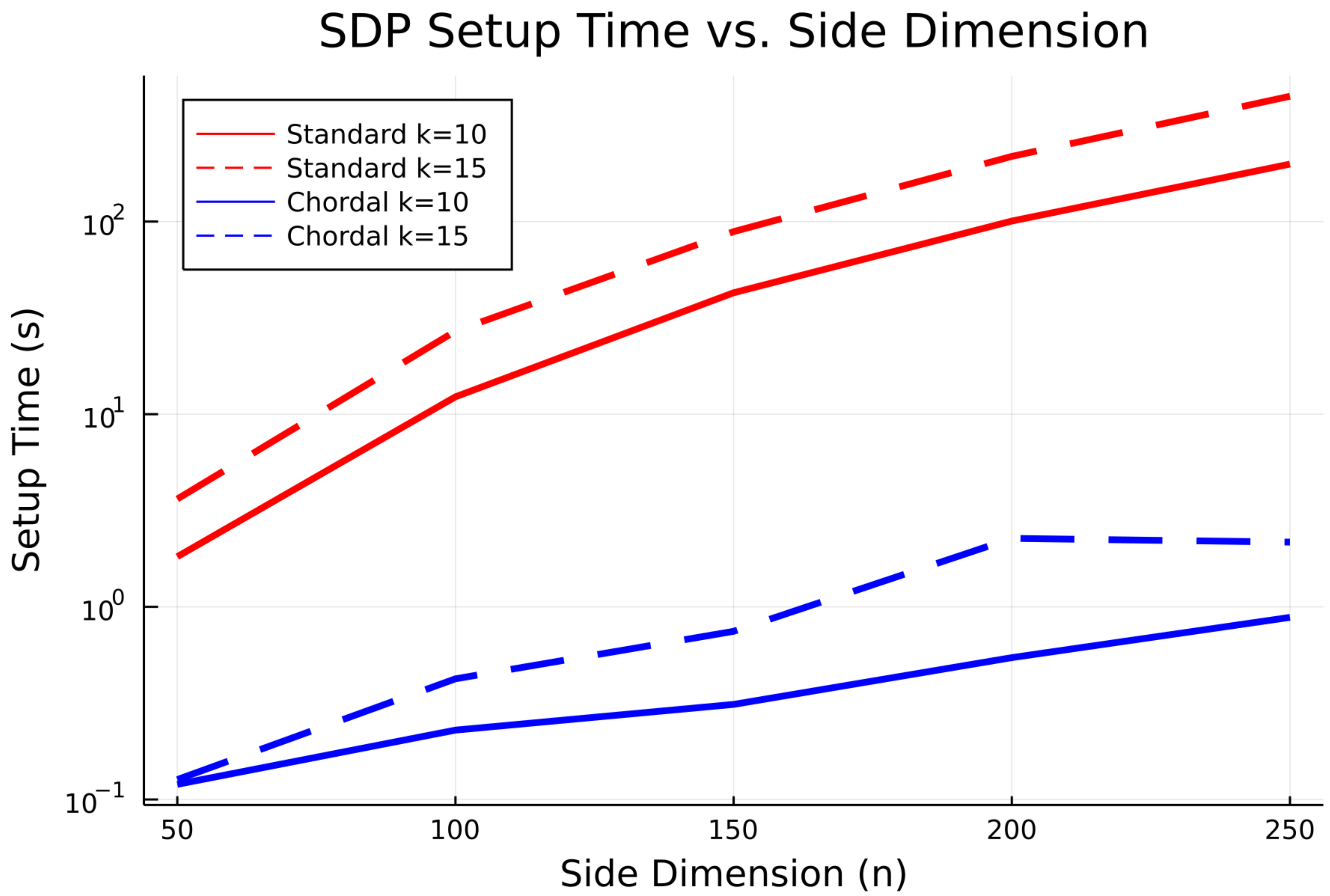
Chordal decomposition *dramatically* speeds up SDPs

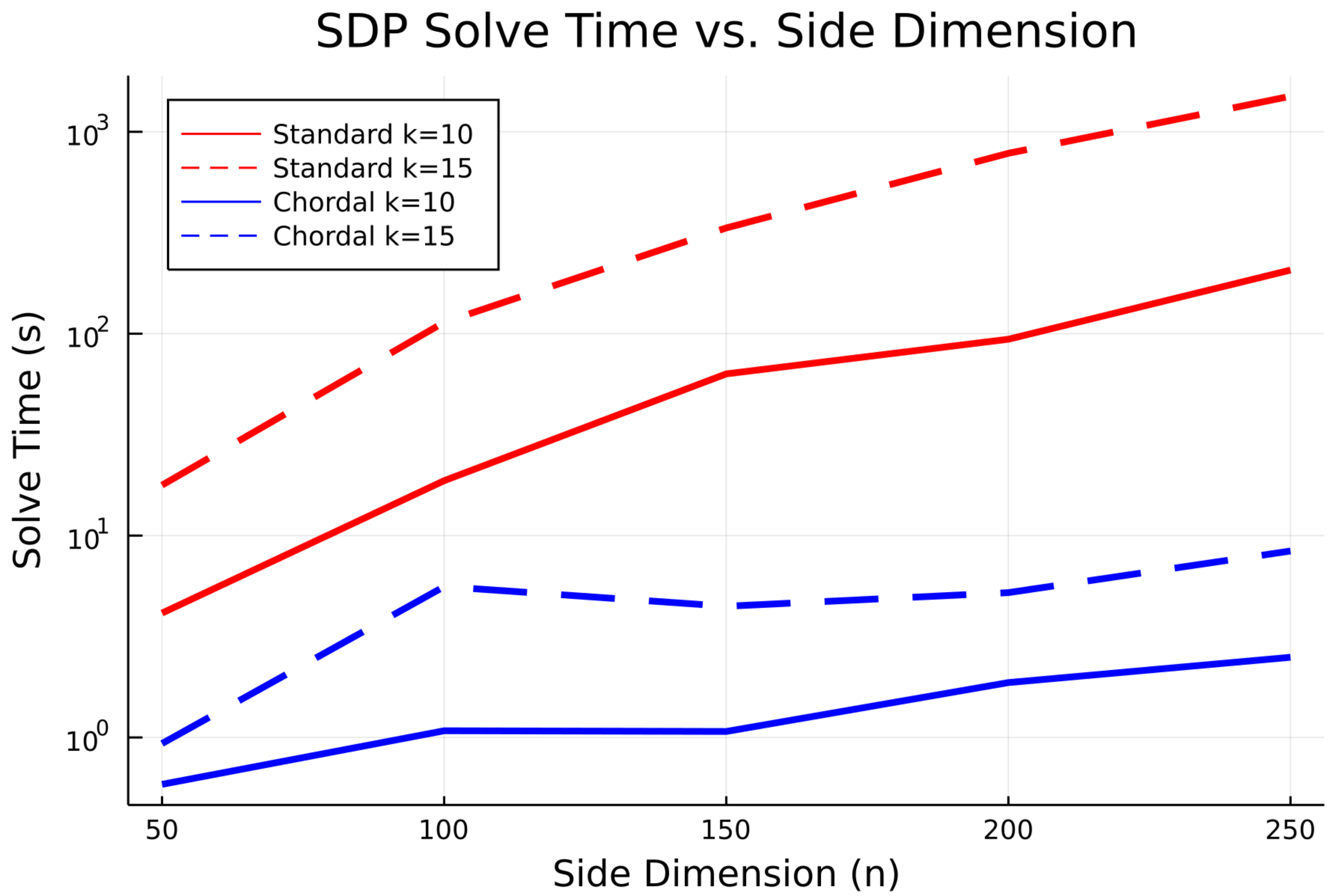




Up to 175x faster on maximum variance unfolding!

Chordal decomposition *dramatically* speeds up SDPs





Up to 175x faster on maximum variance unfolding!

Wrap up

Chordal.jl has several functions for sparse matrices

Survey: [Vandenberghe and Andersen 2015]

- Chordal decomposition of SDPs (with JuMP.jl) [Fukuda et al. 2001]
- Elimination trees, clique trees
- Clique graphs & merging [Garstka et al. 2020]
- Maximum determinant & minimum rank PSD completion [Sun 2015]
- Chordality tests [Tarjan and Yannakakis 1984]
- Euclidean distance matrix completion
- Several examples

Project Ideas

- Use Maximum Variance Unfolding to embed and then classify MNIST
  - Q: Does error propagate up the clique tree? Can we see this in accuracy?
- Investigate scaling of decomposed vs. standard sparse SDPs
  - Big issue in SDP solvers (especially first order ones)
- Implement a pure Julia sparse super nodal Cholesky
- Use these techniques for discrete optimization problems (see Ch 7 in [VA15])
- Integrate with Jump.jl (maybe not for this class, but great open source contribution!)

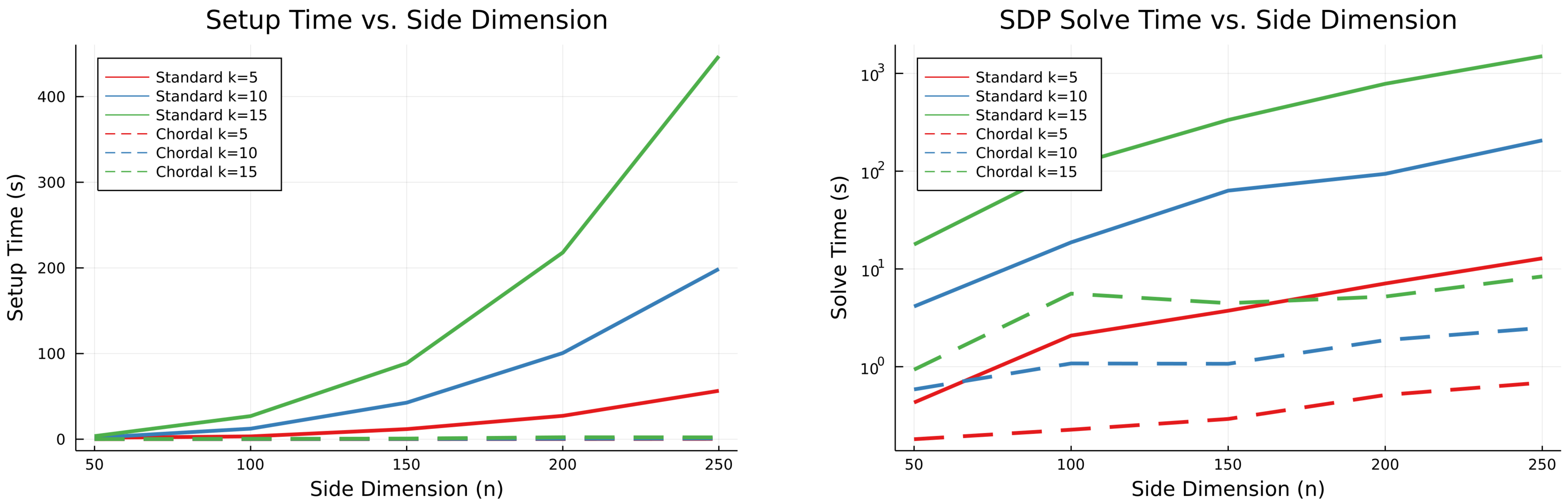
References

- Martin Andersen et al. “Logarithmic barriers for sparse matrix cones.” In: Optimization Methods and Software 28.3 (2013), pp. 396-423
- Mituhiro Fukuda et al. “Exploiting sparsity in semidefinite programming via matrix completion I: General framework.” In: SIAM Journal on Optimization 11.3 (2001), pp. 647–674.
- Michael Garstka, Mark Cannon, and Paul Goulart. “A clique graph based merging strategy for decomposable SDPs.” In: IFAC-PapersOnLine 53.2 (2020), pp. 7355–7361.
- Yifan Sun. “Decomposition methods for semidefinite optimization.” PhD thesis. UCLA, 2015.
- Robert E Tarjan and Mihalis Yannakakis. “Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs.” In: SIAM Journal on computing 13.3 (1984), pp. 566–579.
- Lieven Vandenberghe and Martin S Andersen. “Chordal graphs and semidefinite optimization.” In: Foundations and Trends in Optimization 1.4 (2015), pp. 241–433.

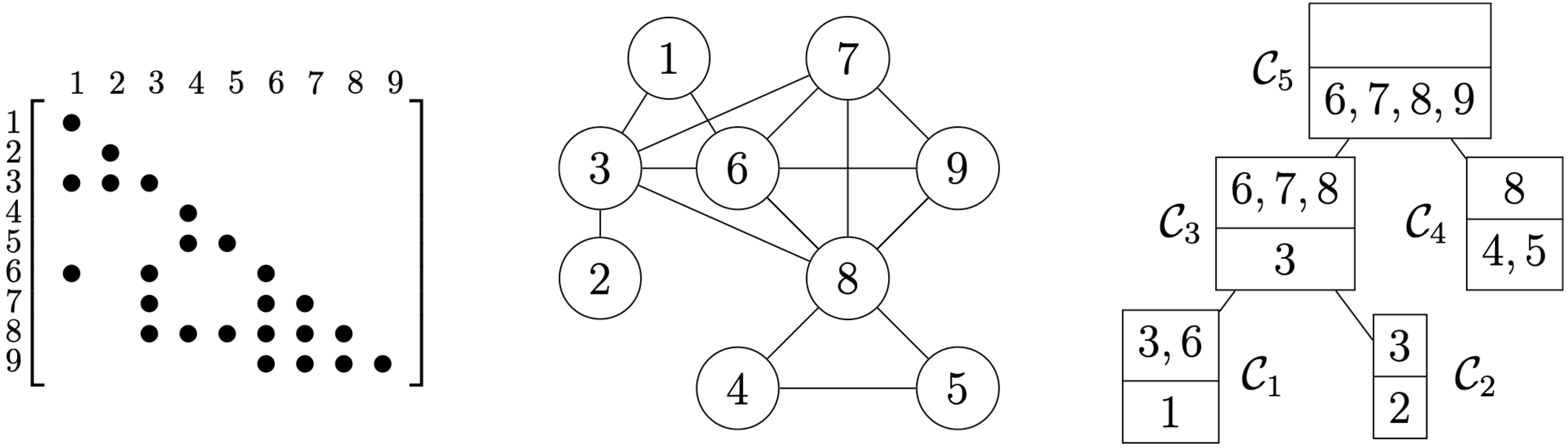
Appendix

Chordal decomposition *dramatically* speeds up SDPs

(Primal form SDP)



Up to 200x faster on maximum variance unfolding!


1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
1 7 3 6 9 2 8 4 5
𝒞5
6, 7, 8, 9
𝒞3
6, 7, 8
3
𝒞4
8
4, 5
3, 6
1
𝒞1
3
2
𝒞2

Definition 4.1 (Running intersection property)**.**
For each pair of cliques $\mathcal{C}_i, \mathcal{C}_j \in \mathcal{B}$, the intersection $\mathcal{C}_i \cap \mathcal{C}_j$ is contained in all the cliques on the path in the clique tree connecting $\mathcal{C}_i$ and $\mathcal{C}_j$.

This property is also referred to as the *clique-intersection property* in [NFF+03] and the *induced subtree property* in [VA+15]. For a given chordal graph, a clique tree can be computed using the algorithm described in [PS90]. The clique tree for an example sparsity pattern is shown in Figure 1(c).

In a *post-ordered* clique tree the descendants of a node are given consecutive numbers, and a suitable post-ordering can be found via depth-first search. For a clique $\mathcal{C}_\ell$ we refer to the first clique encountered on the path to the root as its *parent clique* $\mathcal{C}_{\text{par}}$. Conversely $\mathcal{C}_\ell$ is called the *child* of $\mathcal{C}_{\text{par}}$. If two cliques have the same parent clique we refer to them as *siblings*.