The Journal of Open Source Software

MatrixBandwidth.jl: Fast algorithms for matrix bandwidth minimization and recognition

Luis M. B. Varona [1,2,3]

1 Department of Politics and International Relations, Mount Allison University, Canada **2** Department of Mathematics and Computer Science, Mount Allison University, Canada **3** Department of Economics, Mount Allison University, Canada

DOI: 10.21105/joss.09136

Software

- Review
- Repository
- Archive

Editor: Mehmet Hakan Satman

Reviewers:

- @lucaferranti
- @tmigot

Submitted: 25 September 2025
Published: 11 December 2025

Summary

The *bandwidth* of an $n \times n$ matrix A is the minimum non-negative integer $k \in \{0, 1, \ldots, n-1\}$ such that $A_{i,j} = 0$ whenever $|i - j| > k$. Reordering the rows and columns of a matrix to reduce its bandwidth has many practical applications in engineering and scientific computing: it can improve performance when solving linear systems, approximating partial differential equations, optimizing circuit layout, and more (Mafteiu-Scai, 2014). There are two variants of this problem: *minimization*, which involves finding a permutation matrix P such that the bandwidth of PAP^{T} is minimized, and *recognition*, which entails determining whether there exists a permutation matrix P such that the bandwidth of PAP^{T} is less than or equal to some fixed non-negative integer (an optimal permutation that fully minimizes the bandwidth of A is not required). Accordingly, MatrixBandwidth.jl offers fast algorithms for matrix bandwidth minimization and recognition. Julia's (Bezanson et al., 2017) combination of easy syntax and high performance, along with its rapidly growing ecosystem for scientific computing, made it the ideal language of choice for this project.

Example

Consider the following 60×60 sparse matrix with initial bandwidth 51:

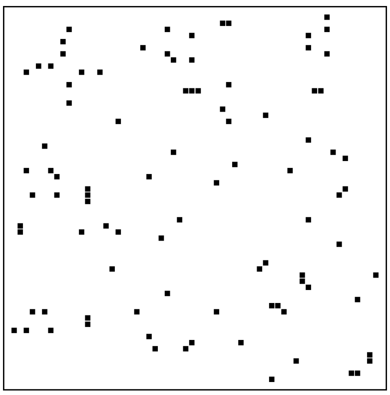

Figure 1: Original 60×60 matrix with bandwidth 51

MatrixBandwidth.jl can both recognize whether the minimum bandwidth of A is less than or equal to some fixed integer (Figure 2) and actually minimize the bandwidth of A (Figure 3):

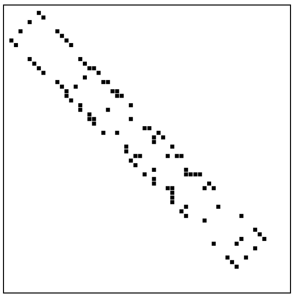

Figure 2: The matrix with bandwidth recognized as ≤ 6 via the Del Corso–Manzini algorithm

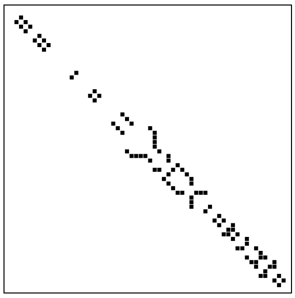

Figure 3: The matrix with bandwidth minimized to 5 via the Gibbs–Poole–Stockmeyer algorithm

Note that since Gibbs–Poole–Stockmeyer is a heuristic algorithm, 5 may not be the *true* minimum bandwidth of A, but it is likely close.

Algorithms

The following matrix bandwidth reduction algorithms are currently available:

- Minimization
  - Exact
    - Caprara–Salazar-González (Caprara & Salazar-González, 2005)
    - Del Corso–Manzini (Del Corso & Manzini, 1999)
    - Del Corso–Manzini with perimeter search (Del Corso & Manzini, 1999)
    - Saxe–Gurari–Sudborough (Gurari & Sudborough, 1984; Saxe, 1980)
    - Brute-force search
  - Heuristic
    - Gibbs–Poole–Stockmeyer (Gibbs et al., 1976)
    - Cuthill–McKee (Cuthill & McKee, 1969)
    - Reverse Cuthill–McKee (Cuthill & McKee, 1969; George, 1971)
- Recognition
  - Caprara–Salazar-González (Caprara & Salazar-González, 2005)
  - Del Corso–Manzini (Del Corso & Manzini, 1999)
  - Del Corso–Manzini with perimeter search (Del Corso & Manzini, 1999)
  - Saxe–Gurari–Sudborough (Gurari & Sudborough, 1984; Saxe, 1980)
  - Brute-force search

Recognition algorithms determine whether any row-and-column permutation of a matrix induces bandwidth less than or equal to some fixed integer. Exact minimization algorithms always guarantee optimal orderings to minimize bandwidth, while heuristic minimization algorithms produce near-optimal solutions more quickly. Metaheuristic minimization algorithms employ iterative search frameworks to find better solutions than heuristic methods (albeit more slowly); no such algorithms are already implemented, but several (e.g., simulated annealing) are currently under development.

Thus far, the Caprara–Salazar-González algorithms are the only ones implemented that require integer linear programming; it is for these that the JuMP.jl package (Lubin et al., 2023) is included as a dependency.

Statement of need

Many matrix bandwidth reduction algorithms exist in the literature, but implementations in the open-source ecosystem are scarce, with those that do exist primarily tackling older, less efficient algorithms. The Boost libraries in C++ (Lumsdaine et al., 2001), the NetworkX library in Python (NetworkX Developers, 2025), and the MATLAB standard library (MATLAB Developers, 2025) all only implement the aforementioned reverse Cuthill–McKee algorithm from 1971. In Julia, the only other relevant packages identified by the author are BandedMatrices.jl (JuliaLinearAlgebra Developers, 2016) and SymRCM.jl (Krysl, 2020), both of which also only implement reverse Cuthill–McKee as their sole bandwidth reduction algorithm.

Furthermore, not enough attention is given to recognition algorithms or exact minimization algorithms. Although more performant modern alternatives are often neglected, at least reverse Cuthill–McKee is a widely implemented method of approximating a minimal bandwidth ordering (as noted above). However, no such functionality for recognition or exact minimization is widely available, requiring researchers with such needs to fully re-implement these algorithms themselves.

These two gaps in the ecosystem not only make it difficult for researchers to benchmark and compare new proposed algorithms but also preclude the application of the most performant modern algorithms in real-life industry settings. MatrixBandwidth.jl aims to bridge this gap by presenting a unified interface for matrix bandwidth reduction algorithms in Julia.

Research applications

The author either has used or is using MatrixBandwidth.jl to do the following:

- Develop a new polynomial-time algorithm for "bandwidth $\leq k$" recognition efficient for both small and large k, and benchmarking it against other approaches (Gurari & Sudborough, 1984; Saxe, 1980)
- Speed up k-coherence checks of quantum states in many cases by confirming that the density matrix's minimum bandwidth is greater than k (Johnston et al., 2025)
- Compute the spectral graph property of "S-bandwidth" (Johnston & Plosker, 2025) via the SDiagonalizability.jl package (Varona et al., 2025), which depends critically on MatrixBandwidth.jl for bandwidth recognition
- Investigate the precise performance benefits of reducing the propagation graph's bandwidth when training a recurrent neural network, building on Balog et al. (2019)

The first three use cases rely on the recognition and exact minimization functionality unique to MatrixBandwidth.jl (indeed, they largely motivated the package's development). The last (ongoing) research project *could* be facilitated by SymRCM.jl instead, but the author intends to use more performant metaheuristic minimization algorithms currently under development when producing the final computational results, as well as use recognition algorithms to minimize bandwidth to various target levels when quantifying performance improvements.

Limitations

Currently, MatrixBandwidth.jl's core functions generically accept any input of the type `AbstractMatrix{<:Number}`, not behaving any differently when given sparsely stored matrices (e.g., from the SparseArrays.jl standard library package). Capabilities for directly handling graph inputs (aiming to reduce the matrix bandwidth of a graph's adjacency) are also not available. Given that bandwidth reduction is often applied to sparse matrices and graphs, this will be addressed in future releases.

Moreover, many of the algorithms only apply to structurally symmetric matrices (i.e., those whose nonzero pattern is symmetric). However, this is a limitation of the algorithms themselves, not the package's implementation. Future releases with metaheuristic algorithms will include more methods that accept structurally asymmetric inputs.

Conflict of interests

The author declares no conflict of interest.

Acknowledgements

I owe much to my research supervisors—Nathaniel Johnston, Sarah Plosker, and Craig Brett—for supporting and guiding me in my work. I would also like to thank Liam Keliher, Peter Lelièvre, and Marco Cognetta for useful discussions. Finally, credit for MatrixBandwidth.jl's telepathic-cat-and-turtle logo goes to Rebekka Jonasson.

References

Balog, M., Merriënboer, B. van, Moitra, S., Li, Y., & Tarlow, D. (2019). *Fast training of sparse graph neural networks on dense hardware*. https://doi.org/10.48550/arXiv.1906.11786

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. https://doi.org/10.1137/141000671

Caprara, A., & Salazar-González, J.-J. (2005). Laying out sparse graphs with provably minimum bandwidth. *INFORMS Journal on Computing*, *17*(3), 356–373. https://doi.org/10.1287/ijoc.1040.0083

Cuthill, E., & McKee, J. (1969). Reducing the bandwidth of sparse symmetric matrices. *Proceedings of the 24th National Conference of the ACM*, 157–172. https://doi.org/10.1145/800195.805928

Del Corso, G. M., & Manzini, G. (1999). Finding exact solutions to the bandwidth minimization problem. *Computing*, *62*, 189–203. https://doi.org/10.1007/s006070050002

George, J. A. (1971). *Computer implementation of the finite element method* [PhD thesis, Department of Computer Science, Stanford University]. https://apps.dtic.mil/sti/tr/pdf/AD0726171.pdf

Gibbs, N. E., Poole, W. G., Jr., & Stockmeyer, P. K. (1976). An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM Journal on Numerical Analysis*, *13*(2), 236–250. https://doi.org/10.1137/0713023

Gurari, E. M., & Sudborough, I. H. (1984). Improved dynamic programming algorithms for bandwidth minimization and the MinCut Linear Arrangement problem. *Journal of Algorithms*, *5*(4), 531–546. https://doi.org/10.1016/0196-6774(84)90006-3

Johnston, N., Moein, S., & Plosker, S. (2025). The factor width rank of a matrix. *Linear Algebra and Its Applications*, *716*, 32–59. https://doi.org/10.1016/j.laa.2025.03.016

Johnston, N., & Plosker, S. (2025). Laplacian $\{-1,0,1\}$- and $\{-1,1\}$-diagonalizable graphs. *Linear Algebra and Its Applications*, *704*, 309–339. https://doi.org/10.1016/j.laa.2024.10.016

JuliaLinearAlgebra Developers. (2016). *BandedMatrices.jl: A Julia package for representing banded matrices*. GitHub. https://github.com/JuliaLinearAlgebra/BandedMatrices.jl

Krysl, P. (2020). *SymRCM: Reverse Cuthill-McKee node-renumbering algorithm for sparse matrices*. GitHub. https://github.com/PetrKryslUCSD/SymRCM.jl

Lubin, M., Dowson, O., Dias Garcia, J., Huchette, J., Legat, B., & Vielma, J. P. (2023). JuMP 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, *15*, 581–589. https://doi.org/10.1007/s12532-023-00239-3

Lumsdaine, A., Lee, L.-Q., Siek, J. G., Gregor, D., & McGrath, K. D. (2001). *Example/cuthill_mckee_ordering.cpp*. Boost v1.37.0 documentation. https://www.boost.org/doc/libs/1_37_0/libs/graph/example/cuthill_mckee_ordering.cpp

Mafteiu-Scai, L. O. (2014). The bandwidths of a matrix. A survey of algorithms. *Annals of West University of Timisoara - Mathematics and Computer Science*, *52*(2), 183–223. https://doi.org/10.2478/awutm-2014-0019

MATLAB Developers. (2025). *Symrcm - sparse reverse Cuthill-McKee ordering - MATLAB*. MATLAB R2025b documentation. https://www.mathworks.com/help/matlab/ref/symrcm.html

NetworkX Developers. (2025). *Source code for networkx.utils.rcm*. NetworkX v3.5 documentation. https://networkx.org/documentation/stable/_modules/networkx/utils/rcm.html

Saxe, J. B. (1980). Dynamic-programming algorithms for recognizing small-bandwidth graphs in polynomial time. *SIAM Journal on Algebraic and Discrete Methods*, *1*(4), 363–369. https://doi.org/10.1137/0601042

Varona, L. M. B., Johnston, N., & Plosker, S. (2025). *SDiagonalizability: A dynamic algorithm to minimize or recognize the* s*-bandwidth of an undirected graph*. GitHub. https://github.com/GraphQuantum/SDiagonalizability.jl